SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Dejan Starčević

Zagreb, 2013.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Bojan Jerbić

Student:

Dejan Starčević

Zagreb, 2013.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem se svojim mentorima prof. dr. sc. Bojanu Jerbiću i dr. sc. Tomislavu Stipančiću sa Fakulteta strojarstva i brodogradnje na pruženoj stručnoj pomoći, motivaciji i savjetima prilikom izrade Diplomskog rada.

Zahvaljujem se kolegama sa Fakulteta strojarstva i brodogradnje, prijateljima i dragim osobama na moralnoj podršci te koji su protekle studentske dane učinili znatno ljepšim.

Posebno se zahvaljujem svojim roditeljima i sestri na razumijevanju, nesebičnoj podršci i na žrtvi koju su morali podnijeti za vrijeme studiranja dok sam bio fizički, odnosno psihički nedostupan.

Dejan Starčević

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

Središnje povjerenstvo za završne i diplomske ispite

Povjerenstvo za diplomske ispite studija strojarstva za smjerove:

procesno-energetski, konstrukcijski, brodostrojarski i inženjersko modeliranje i računalne simulacije

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: Mat. br.:

Naslov rada na hrvatskom jeziku:

Naslov rada na engleskom jeziku:

Opis zadatka:

Zadatak zadan: Rok predaje rada: Predviđeni datumi obrane:

Zadatak zadao: Predsjednik Povjerenstva:

Prof. dr. sc. Zvonimir Guzović

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA I KRATICA

Oznaka/kratica	Jedinica	Opis, značenje
AAA	-	Anyone can say Anything about Anything
AL	-	Assembly Line (Linija za sklapanje)
ARC2	-	Appmosphere RDF Classes 2
ARPANET	-	Advanced Research Projects Agency Network
BP	-	Behavioral Pattern (obrazac ponašanja)
CERN	-	European Organization for Nuclear Research
CS	-	Capacitive Sensor (kapacitivni senzor)
CSS	-	Cascading Style Sheets
DL	-	Descriptive Logic (deskriptivna logika)
DAML	-	DARPA Agent Markup Language
DARPA	-	Defense Advanced Research Projects Agency
GUI	-	Graphical User Interface
HTML	-	HyperText Markup Language
HTTP	-	Hypertext Transfer Protocol
IDE	-	Integrated Development Environment
IP	-	Internet Protocol
LES	-	Live Enterprise Server
MAS	-	MultyAgent System
ME	-	Maestro Edition
MVC	-	Model View Controller
MySQL	-	My Structured Query Language
NS	-	NameSpace
OIL	-	Ontology Interface Layer
OWA	-	Open World Assumption
OWL	-	Ontology Web Language
PHP	-	PHP: Hypertext Preprocessor
RDF	-	Resource Description Framework
RDFS	-	RDFShema
RIF	-	Rule Interchange Format
RML	-	Rule Markup Language
SP	-	Stopping Place (zaustavno mjesto)
SPARQL	-	SPARQL Protocol and RDF Query Language
SQL	-	Structured Query Language
SWP	-	SPARQL Web Pages

SWRL	-	Semantic Web Rule Language
TBC	-	TopBraid Composer
TBS	-	TopBraid Suit (TopBraid paket alata)
TC1	-	Tool Changer 1 (izmjenjivač alata 1)
TC2	-	Tool Changer 2 (izmjenjivač alata 2)
TCP	-	Transmission Control Protocol
URI	-	Uniform Resource Information
URL	-	Uniform Resource Locator
VS1	-	Vision Sensor 1 (vizijski senzor 1)
VS2	-	Vision Sensor 2 (vizijski senzor 2)
W3C	-	WWW consortium (WWW konzorcij)
WAMP	-	Windows Apache MySQL PHP
WP	-	Working Place (radno mjesto)
WWW	-	World Wide Web
XML	-	Extensible Markup Language

SAŽETAK

Ideja semantičkog weba, uz stvaranje globalne mreže znanja, je i definirnje značenja za svaki podatak kako bi računala mogala razumjeti značenje podataka, a ne samo strukturu tih podataka. Sustavi koji koriste koncepte sveprisutnog računarstva uz navedene povezane podatke mogu autonomno predviđati ponašanje u specifičnim situacijama. Predviđanje ponašanja vrši se na temelju informacija prikupljenih iz okoline, pomoću raznih senzora, koje se spremaju u ontologiju. Ontologije predstavljaju model za prikupljanje, pohranu i predstavljanje znanja o promatranoj domeni, a zasnivaju se na identifikaciji koncepcije te na definiranju relacija između njih.

U okviru rada opisana je ideja i tehnologija za povezivanje podataka i razvoj semantičkog weba. Primjenom alata *TopBraid Composer* izrađen je primjereni ontološki model po uzoru na robotski montažni sustav u Laboratoriju za projektiranje izradbenih i montažnih sustava, Fakulteta strojarstva i brodogradnje. Na temelju izrađenog ontološkog modela razvijen je način za predlaganje obrazca ponašanja robota u ovisnosti o ulaznim vrijednostima. Korišteni mehanizmi predlaganja temeljeni su na deskriptivnoj logici ostvarenoj unutar ontologije. Za pohranu i kasniju eksploataciju ontologije razvijeno je korisničko sučelje pomoću *TopBraid* alata i web aplikacija na temelju *ARC2* sustava.

Ključne riječi: ARC2 sustav, ontologije, OWL, RDF, robotski montažni sustavi, semantički web, SPARQL, TopBraid

SUMMARY

The idea of semantic web, besides creating global network of mutually linked data, is to define the meaning of each piece of data in order for the machines to understand the content as well as the structure of that data. Systems that use the concept of ubiquitous computing and linked data can autonomously estimate behavioral patterns in specific situations. Estimation of the behavioral patterns is provided on the basis of information collected from the environment using a varity of sensors; information is then stored in the ontology. Ontology is a model for collecting, storing and presentation of knowledge about the observed domain, it is based on the concept of identifying and defining the relationship between them.

This paper presents the ideas and technologies for developing linked data applications and semantic web. Along the lines of robotic assembly system at the Laboratory for design production and assembly systems, Faculty of Mechanical Engineering, an appropriate ontology model was made with the use of TopBraid Composer. Based on the created ontological model a system to propose the behavioral patterns of a robot depending on the input values was developed. Proposing mechanisms used are based on the descriptive logic realized within the ontology. User interface and web applicaton for storage and subsequent exploitaton of ontology was developed using TopBraid tools and ARC2 system.

Key words: ARC2 system, ontology, OWL, RDF, robotic assembly systems, semantic web, SPARQL, TopBraid

1. UVOD

Mreža informacija je prije par desetljeća bila tehnička ideja dostupna samo visoko obrazovanim individualcima i eliti profesionalaca informacijske tehnologije. Danas je normalno očekivati da su gotovo svi upoznati s osnovnom idejom i pojmovima vezanima uz mrežu informacija rasprostranjenu po cijelome svijetu, odnosno s internetom. Internet je prepun podataka i aplikacija koje nam svakodnevno služe za brže i efikasnije snalaženje. Zbog velike količine podataka često se zna doći do krivih, netočnih ili nepotrebnih podataka i dokumenata. Razlog tome je nedosljednost, nepovezanost i slaba pristupačnost određenih podataka. Da bi se takve pogreške ispravile, uvela se semantička tehnologija pomoću koje se određeni podaci dovode u međusobni odnos, tj. podaci se povezuju jedan s drugim i tako čine smislenu cjelinu određene namjene. Ideja povezanih podataka (Linked Data) temelji se na ideji semantičkog weba gdje podaci ne bi bili vezani samo lokacijski već bi se podacima pridodao smisao i značenje.

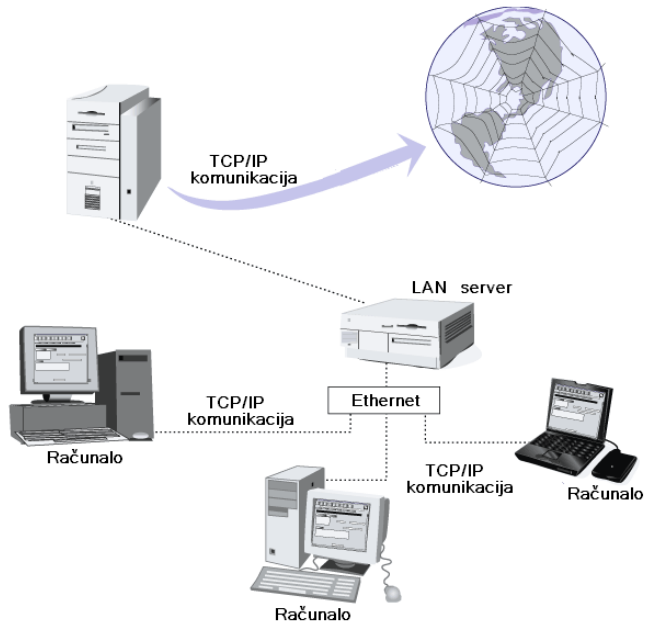
Povezivanje podataka vrši se pomoću ontoloških jezika i pomoću programa za izradu ontologija. Ontologije predstavljaju model za prikupljanje, pohranu i predstavljanje znanja o promatranoj domeni, a zasnivaju se na identifikaciji koncepcija te na definiranju relacija između njih. U ovisnosti o problematici koja se opisuje, ulaznim stanjima te implementiranim logičkim relacijama, ontologije omogućavaju kreiranje spoznaja o različitim aspektima domene problema. Ontologije se danas koriste ne samo u webu (semantičkom webu), već i u umjetnoj inteligenciji, software inženjeringu i informacijskoj arhitekturi kao oblik reprezentacije znanja o okolini ili nekom njezinom dijelu.

U proizvodnim procesima su sve prisutniji robotski sustavi te se ovom radu ontologija uz prikladnu tehnologiju koristi i pri upravljanju sustavima robotskog sklapanja. Svaki element robotskog sustava za sklapanje u ontologiji je predstavljen kao klasa ili instanca te su mu dodijeljena svojstva i mjesto u hijerarhiji sustava. Time je element uspješno „preslikan“ iz stvarnog svijeta u računalni. Izrađeni su svi ontološki modeli, dodijeljene su im vrijednosti i međusobni odnos te kao takvi su pohranjeni i spremni za kasniju eksploataciju. Eksploatacija ontoloških modela vrši se pomoću posebno izrađenih web aplikacija koje omogućavaju da se modelu pristupi s bilo kojeg računala, čime se omogućuje potpuna fleksibilnost pri iskorištavanju razvijenog ontološkog modela.

1.1. Razvoj weba

Internet je globalni informacijski sustav logički povezan jedinstvenim sustavom adresiranja putem internet protokola (TCP/IP) koji povezuje računala i računalne mreže. Internet je nastao 1969. godine u SAD-u u sklopu istraživanja američkog Ministarstva obrane. Zvao se ARPANET prema agenciji za napredne istraživačke projekte (*Advanced Research Project Agency*) [5]. Pridonositelji popularnosti mreže su razni i sve zamislive teme su pokrivene. Jedna od najkorištenijih usluga interneta je *World Wide Web*. Naziv dolazi iz engleskog jezika i može se prevesti kao „svjetska mreža“ (skraćeno se može pisati *WWW, W3* ili samo *Web*) i služi za dohvaćanje hipertekstualnih dokumenata koji mogu sadržavati tekst, slike i razne multimedijalne sadržaje za čiji se prikaz koriste računalni programi koji se nazivaju web-preglednici.

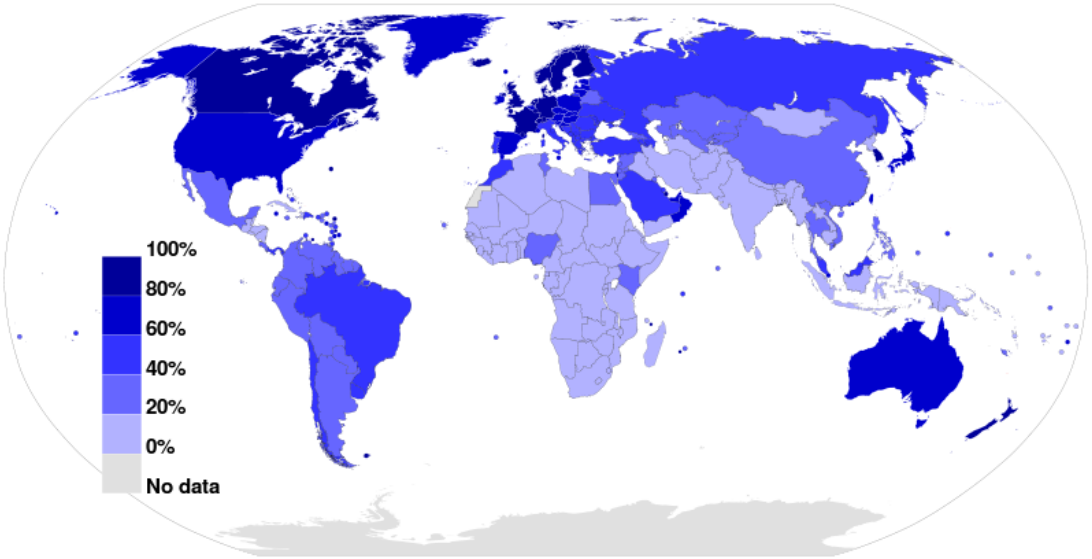
Projekt kojim je počeo razvoj weba predložio je 1990. godine u CERN-u engleski inženjer i znanstvenik Tim Berners-Lee. U sklopu weba postoje i standardi koji predstavljaju skup smjernica i preporuka za ispravno kreiranje HTML, CSS i XML koda s ciljem nesmetanog i jednostavnijeg pristupa internet tenologijama. Međusobno povezivanje dvaju ili više računala koji dijele neke resursa (podatke, sklopovlje, programe) naziva se računalna mreža [Slika 1.]. Za međusobnu komunikaciju računala koriste komunikacijske protokole.



Slika 1. Računalna mreža

Tijelo koje nadgleda razvoj standarda i brine se da stranice izrađene u tom obliku budu što dostupnije korisnicima je *World Wide Web Consortium (W3C).* Važne osobine trenutnog weba (sintaktičnog weba) su njegova pristupačnost i jednostavnost. Ideja weba je jednostavna, a to je da se razvije svijet informacija kojem svatko može doprinijeti svojim idejama i tako upotpuniti određenu cjelinu kojoj svatko može pristupiti. Upravo je takva otvorenst weba najzaslužniji faktor njegovoj začuđujućoj sveobuhvatnosti pokrivenih tema. Mreža informacija „web" je mješavina teksta, prezentacija, multimedije, analiza i svih ostalih oblika informacije koje je određena osoba voljna iz vlastitih ili tuđih potreba ili zabave objaviti putem web stranica. Web kao takva mješavina je veoma koristan te je svatko s malo strpljenja i razumjevanja u stanju domoći se potrebnih informacija. Često se za sintaktički web kaže da je veoma rasprostranjen ali plitak, što znači da iako na webu postoji mnogo informacija, one su nedosljedne, neintegrirane i slabo povezane.

Zbog takve nedosljednosti weba moguće je za traženi pojam pronaći više odgovora od kojih neki i ne moraju biti istiniti. Kako u takvom slučaju razlikovati točne od netočnih podataka, odnosno potrebne od nepotrebnih informacija? Moguće je (pošto svatko može doprinjeti širenju i nadopunjavanju weba po vlastitoj volji) da za određeni podatak postoji više informacija, nemoguće je znati jesu li sve informaciji dosljedne, niti u kakvom su međusobnom odnosu. Isto tako je teško doći do potpunih informacija bez dodatnog pretraživanja weba.



Slika 2. Dostupnost interneta po svijetu

Danas je web pun raznih inteligentnih integriranih aplikacija koje su prije izgledale „futuristički“. Tražilice nam daju pametne, intuitivne i personalizirane odgovore na naš upit, razne stranice imaju integrirane karte za lakše snalaženje, stranice sa multimedijom imaju mogućnost postavljanja vlasitih lista za pregledavanje. Nove inovacije u sklopu web stranica razvijaju se svakodnevno. Gotovo svaku informacijsku tehnologiju moguće je implementirati u sklopu web stranice. S preko bilijun stranica (često se jedna web stranica sastoji od više stranica) i milijardu korisnika, web je jedna od najuspješnijih inženjerskih tehnologija ikad napravljenih. Krajem 2009. godine postojalo je 234 milijuna web stranica od kojih je 47 milijuna napravljeno te godine [1]. Web je trenutno veoma bogato skladište multimedijalnih sadržaja: *YouTube* mjesečno ima oko milijardu korisnika koji pregledaju preko 4 milijardi sati video sadržaja [2], dok *Facebook* ima oko 900 milijuna aktivnih korisnika od kojih 50% svakodnevno koristi njegove usluge [3]. Internet je raprostranjen po cijelom svijet i život bez njega danas je nezamisliv. Interenet se uvukao u naše domove te se koristi za zabavu, informiranje i obavljanje raznih poslova.

Količina podataka na webu je zapanjujuća i raste iz dana u dan. Način na koji korisnici pretražuju web sadržaj je preko tražilica uz pomoć ključnih riječi. Svakim danom nastaje sve više pametnih aplikacija te je infrastrukturu weba potrebno prilagoditi kako bi mogao podržati nadolazeće aplikacije. Dakle, kako bi se mogle obuhvatiti sve tehnologije, sadašnje i buduće, potrebno je da infrastruktura weba to može podržati. Iako se čini da će se postići bolje performanse weba ukoliko se postigne inteligentnija infrastruktura weba, to nije točno. Izgradnja inteligentnije infrastrukture weba nije praktična niti je moguća, te se ne može očekivati da će web podržavati apsolutno sve tehnologije. Inteligentnije ponašanje weba se stoga može jedino dobiti inteligentnijim aplikacijama, a ne inteligentnijom infrastrukturom weba [4]. Ipak, infrastruktura weba se može poboljšati kako bi se postigao puni potencijal inteligentnih aplikacija. Čak i najpametnija i pronicljiva aplikacija je inteligentna koliko i podaci koji su joj dostupni. Nedosljedni, nepotpuni ili kontradiktorni podaci će svejedno dati zbunjeni, „glupi“ odgovor čak i ako je aplikacija inteligentna. Prema tome, ne treba napraviti inteligentnu infrastrukturu weba, već je potrebno napraviti takvu infrastrukturu uz koju je najlakše integrirati željenu informaciju na web. Potrebno je osigurati dosljednost, točnost i pristupačnost podataka kako bi pametna aplikacija mogla odraditi svoj dio punim potencijalom. Na takav način će i web izgledati „pametnije“ jer će inteligentne aplikacije imati pristup potrebnim podacima.

1.2. Sveprisutno računarstvo

Sveprisutna, ugrađena računala od izuzetne su važnsti za sveopći razvoj. Minijaturizacija računala omogućuje stapanje računala s okolinom i stvaranje pametne okoline. Ta ista svojstva omogućuju i izgradnju umreženih računalnih sustava namijenjenim obradama informacija i stvaranju novih znanja.[7] Pojam sveprisutnog računarstva predstavlja treće poglavlje u evoluciji modernog računarstva.[8] Prvo poglavlje karakterizira uporaba manjeg broja velikih, centralnih i dijeljenih (tzv. *mainframe*) računala na koja su spajani terminali za pristup. Glavna osobina ove ere računarstva je ostvarivanje istovremene interakcije više korisnika s jednim računalom. Drugo poglavlje u razvoju računarstva pripada uporabi, tzv. osobnih računala (*PC* – engl. *Personal Computer*). U sklopu ove ere omogućen je pristup računalima za sve one korisnike koji su ih mogli priuštiti. Navedeni pristup je rezultirao interakcijom korisnik – računalo (tzv. interakcija jedan na jedan). Treće, odnosno posljednje poglavlje pripada eri, tzv. sveprisutnog računarstva (engl. *Pervasive* ili *Ubiquitous Computing* – *Ubicomp*) koje podrazumijeva smještaj računala (uređaja, strojeva, osobnih računala, pametnih telefona i sl.) u okolinu koja se potom konstantno analizira. Budući da se radi o međusobno povezanim uređajima od kojih svaki može imati jednu ili više funkcija, ostvaruje se interakcija korisnik – više računala (tzv. interakcija jedan na više). Koristeći metode koje se temelje na viziji sveprisutnog računarstva, okolina postaje prostor konstantno analiziran uređajima koji mogu primjetiti (osjetiti, detektirati i sl.) po zadanom kriteriju značajne promjene koje mogu potaknuti sustav da reagira u ovisnosti o njegovoj prvotnoj namjeni.

Sveprisutno računarstvo možemo usporediti s virtualnom stvarnošću. Kod virtualne stvarnosti ljudi pokušavaju okolinu modelirati u računalo te tako kreirati prividnu stvarnost okoline. Koncepcija sveprisutnog računarstva je drugačija, računala se integriraju u okolinu i tako postavljena putem različitih senzora prikupljaju informacije i stvaraju vlastiti model okoline i djelovanja. Razvijeni spoznajni model ovom radu je u skladu s vizijom sveprisutnog računarstva. Ontološka jezgra modela koristi sezore (kapacitivni, vizijski itd.) koji su smješteni u okolinu te pomoću dobivenih podataka donosi odluke. Unutar spoznajnog modela, informacije eksplicitne prirode se interpretiraju na način da roboti (računala) mogu "shvatiti" prirodu trenutnog konteksta okoline te donositi odluke koje su u skladu s vjerovanjima stručnjaka koji je razvijao model.[6]

2. SEMANTIČKI WEB

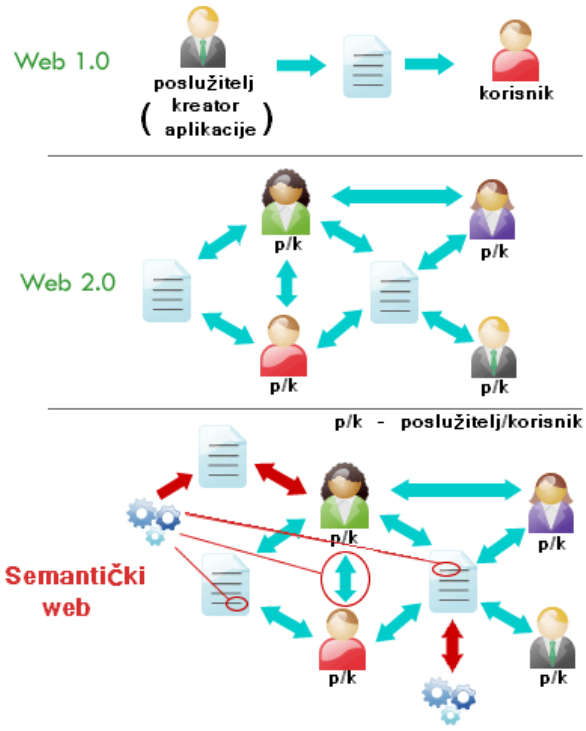
Semantički web nastao je kao rezultat traganja za efikasnijim rješenjima za pronalaženje informacija na webu i predstavlja nastavak, proširenje postojećeg weba. Jedan od bitnih nedostatka sintaktičkog weba je to što su informacije kreirane i prilagođene ljudima i kao takve nisu razumljive računalima. Navedeni problem i problem povezivanja podataka se mogu riješiti semantičkom tehnologijom koja omogućuje smještaj informacija na mreži u formi razumljivoj računalima koja su međusobno povezana s dodijeljenim vrijednostima. Zamišljeno je da semantički web bude ekstenzija sintaktičkog, zadržavajući pritom sva ranije implementirana svojstva, standarde i funkcionalnosti. Semantička tehnologija zasniva se na semantici podataka. Semantika (grčki: *semantikos*, koji daje znakove, značajan, znak) se odnosi na aspekte značenja koji su izraženi u jeziku, odnosno kodu ili nekom drugom obliku predstavljanja.

Semantika se kontrastira dvama aspektima smislenog izraza, napose sintaksom, konstrukcijom složenih znakova iz jednostavnih, te pragmatikom, praktičnoj uporabi znakova u svrhu interpretacije pojedninih okolnosti ili konteksta.[9] Navedena tehnologija omogućuje zajednički rad računala i ljudi proširujući pritom područje primjene mehanizama semantičkog weba na sve umrežene uređaje.

Zadnjih nekoliko godina se intenzivno radilo na prilagodbi, odnosno transformaciji postojećih te izradi novih dokumenata prema standardima *W3* konzorcija.[6] Standard promiče uobičajene tipove podataka na *World Wide Webu* i uključivanje semantičkog sadržaja, Time se želi trenutačna mreža, u kojoj dominira neorganiziranost, provesti u organiziranu mrežu. Semantički web je građen na temelju *W3C*-ovog formata *RDF* (*Resource Description Framework*).

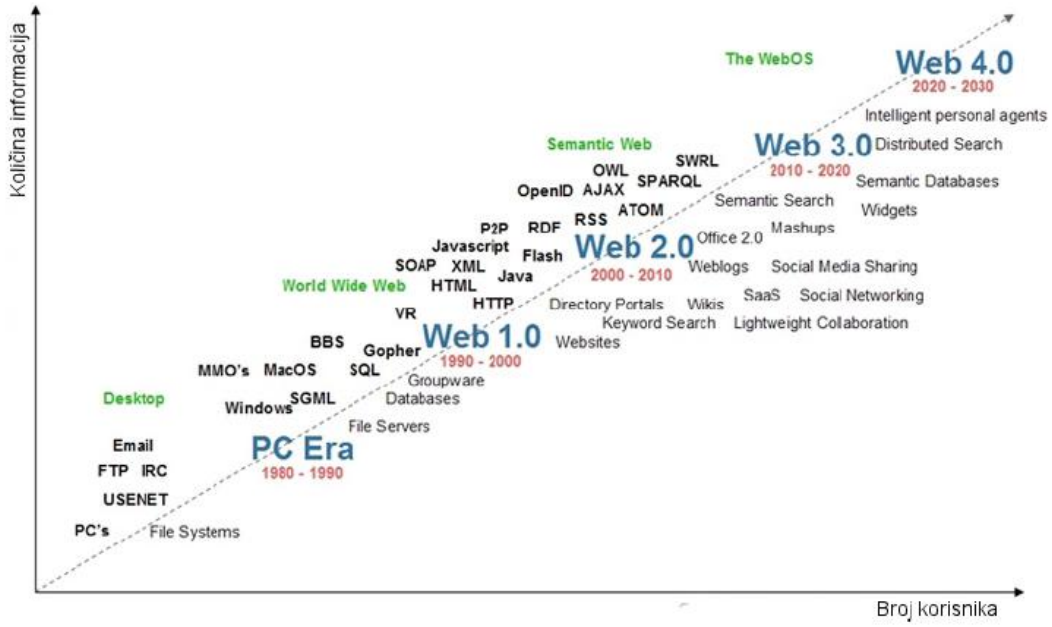
Razvoj weba zasnovan je na tehnološkim i organizacijskim principima: jednostavnost, modularni dizajn, distribucija, decentraliziranost, tolerancija. Upravo su decentralizacija i tolerancija principi koji objedinjuju pravac razvoja semantičkog weba. Ideju za semantički web razvio je Tim Berners-Lee, izumitelj *World Wide Weba*, definirao ga je kao mrežu podataka koju računalo (ili neki drugi uređaj) može razumjeti direktno ili indirektno. Danas su razni izvori informacija i podataka na webu uglavnom namijenjeni ljudskim korisnicima, dok će u budućnosti podatke obrađivati programi kreirani neovisno o podacima.

Programi koji će obrađivati podatke neophodno moraju imati strojno čitljive navode o izvorima informacija i njihovim međusobnim odnosima koji će se temeljiti na postojanju jedinstvenih definicija i na dostupnosti tih definicija programima. Podrazumijeva se da će se internet moći pretraživati ne samo korištenjem riječi, već i kroz upotrebu značenja. Takav način pretraživanja očigledno zahtijeva i semantičku i sintaktičku interoperabilnost predmetnog rječnika, budući da je dobro poznato da su za temeljit i sveobuhvatan opis predmeta potrebni ne samo izolirani pojmovi, već i propozicijska logika. U tom kontekstu, postojeće sustave za organizaciju znanja kao što su, npr. klasifikacijski sustavi, prepoznalo se kao važne izvore strukturiranih i formaliziranih rječnika koji mogu biti izuzetno korisni u razvoju semantičkog weba. Neovisno o korištenom indeksnom pojmu (notacijski simbol ili riječ) sustavi za organizaciju znanja prepoznatljivi su po logičkim procesima koje uključuju, po svojoj strukturi ili prema funkcijama i načinu prikaza znanja. Onima koji rade na primjeni informacijskih sustava od velike su koristi, npr. klasifikacijske strukture koje se mogu upotrijebiti pri izradi informacijskih mapa, pojmovnih mapa, kod vizualizacije pristupa predmetu itd.



Slika 3. Razlika između weba 1.0, weba 2.0 i semantičkog weba

Na slici [Slika 3.] grafički je predočena razlika između weba 1.0, weba 2.0 i semantičkog weba što se tiče čitanja i obrade podataka. Kod weba 1.0 poslužitelj (kreator aplikacije ili web stranice) je postavljao podatke ili dokumente na web koje je samo on mogao mijenjati i orađivati dok su korisnici imali samo uvid u te podatke (*read only*) . Dolaskom weba 2.0 postalo je moguće da svatko ima uvid u postavljene podatke koje ovisno o funkciji mogu mijenjati i obrađivati, time je svatko postao poslužitelj i korisnik. Kod semantičkog weba, uz navedena svojstva weba 2.0 i to što su podaci međusobno povezani, podaci koji su postavljeni na web su i strojno čitljivi te ih računala mogu samostalno mijenjati ili nadopunjivati.

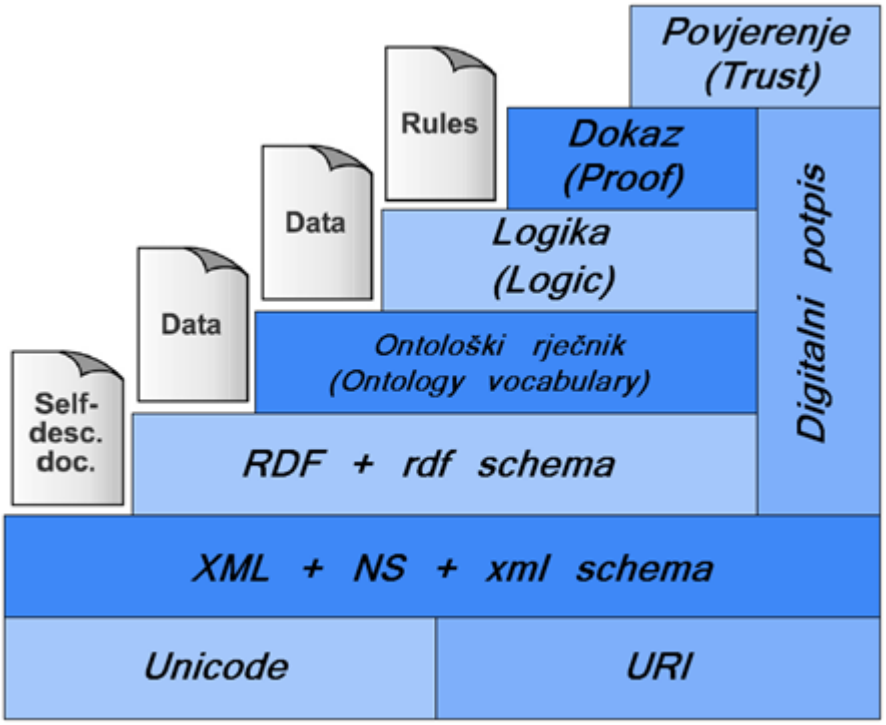


Slika 4. Razvoj weba

Dvije su razvojne tendencije na području sustava za organizaciju znanja, posebice unutar konteksta klasifikacijskih struktura znanja, bitne za razvoj ideje semantičkog weba i vrlo će vjerojatno utjecati na buduću upotrebu sustava za organizaciju znanja. Za razvoj semantičkog weba bitan je razvoj standarda i programskih rješenja za terminološku razmjenu i razvoj ontologijskih rječnika. Uključujući ova dva razvojna područja, proces standardizacije primarno se usredotočuje na tehnološke okvire mreže i na daljnji napredak u razvoju mrežnih jezika za reprezentaciju sadržaja.[10] Primjena semantičkog weba donosi smislenu strukturu web sadržaja gdje su podacima dana precizno definirana značenja stvarajući okruženje za jednostavno korištenje i izvršavanje sofisticiranijih zadataka uspješnijom komunikacijom između računala i korisnika.

2.1. Arhitektura semantičkog weba

Osnovna ideja semantičkog weba je boljom standardizacijom metapodataka (*metainformation*- predstavljaju podatke o podacima, strukturirani podaci koji opisuju, objašnjavaju, lociraju ili a neki drugi način omogućavaju lakše upravljanje resursima) pružiti pomoć korisnicima i inteligentnim software programima u pronalaženju potrebnih podataka na webu. Standardi moraju biti definirani ne samo za sintaktičku formu dokumenata, već i za njihov semantički sadržaj, čime se prtraživačima omogućuje da sami kontaktiraju sve dostupne izvore i pronalaze tražene sadržaje. Bitno je da programi koji se pri tom prilikom koriste, razumiju semantiku informacija koje pretražuju. Do razumijevanja semantike dolazi se korištenjem programskh jezika kojima se određuje semantika podataka i njihovih izvora.

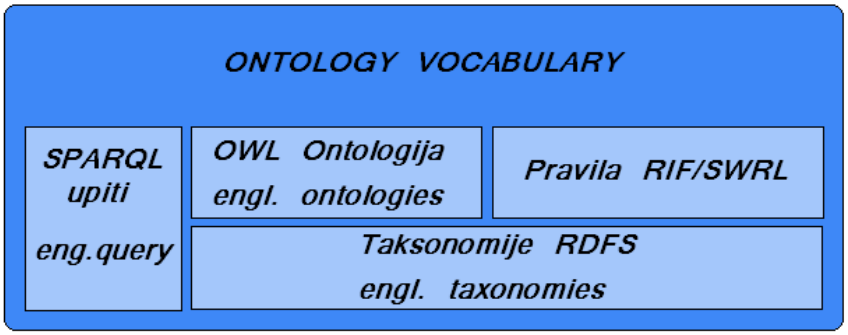


Slika 5. Arhitektura slojeva semantičkog weba

Dok *HTML* (*HyperText Markup language*) omogućava predstavljanje podataka i njihov izgled na webu te opisuje kako neka informacija izgleda na web stranici, arhitekturu semantičkog weba čine dva važna informacijsko-tehnološka standarda. *XML* (*eXtensible Markp Language*) je proširivi jezik za označavanje koji određuje strukturu podataka. *XML* omogućava autorima da kreiraju vlastito označavanj, sintaksu, koji u sebi nosi dio semantike.

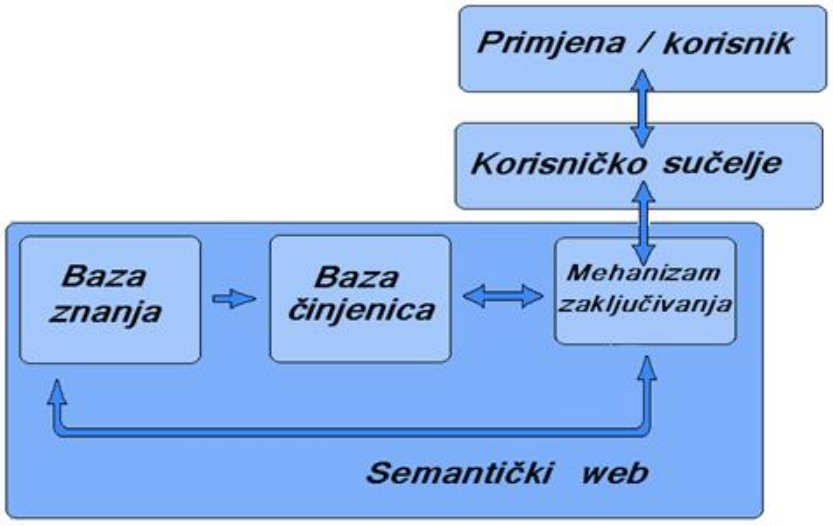
RDF (*Resource Description Framework*) je već spomenut u nekoliko navrata. On je centralni protokol na webu zasnovan kao *W3C* standard, opisuje semantičke veze između elektronskih izvora (resursa). *RDF* omogućuje razvijanje ontološkog sustava koji podržava informacije na webu opisujući semantiku strukture u trojci (*triplet*-u: subjekt, predikat, objekt). Treći važan tehnološko-informacijski standard su ontologije. One su najbitniji i najteže ostvarivi činilac semantičkog weba. Predstavlja skupove pojmova i njihovih međuveza.[15] Na slici [Slika 5.] prikazana je arhitektura sloja semantičkog weba. Prikazani model još se nazva i „*Layer Cake"*. Zadatak mu je standardizirati tehnologiju semantičkog weba a predstavio ga je Tim Berners-Lee. U okvirima prvog sloja nalaze se standardi za prikaz teksta: *Unicode* i *URI* (*Uniform Resource Identifier*). *URI* je niz znakova koji u svrhu identificiranja imena ili web izvora nekog podatka. Ovakav način identifikacije omogućava interakciju putem weba koristeći određene protokole. Svaki *URI* je definiran shemom koja specificira točnu sintaksu i pripadajućim protokolom. Značenje *URI* je šire od značenja *URL* (*Uniform Resource Locator*). Na drugom sloju se nalazi XML i XML Schema koji predstavljau osnovu za postizanje interoperabilnosti na internetu. Na trećem sloju nalaze se *RDF* i *RDF schema*, već spomenuti standardi pri opisivanju metapodataka i konceptnih riječnika na webu. RDF je model koji samo povezuje podatke, a RDF schema je dio koji daje osnovni smisao i načenje vezama te omogućuje hijerarhiju.

U okviru četvrtog sloja arhitekture semantičkog weba nalazi se ontološki rječnik (*Ontology vocabulary*). Navedeni sloj predstavlja sve ontološke jezike koji su napravljeni na bazi *RDF*-a *RDF scheme*. Mogu biti: *OIL* (*Ontology Interface Layer*), *DAML + OIL* (*DARPA Agent Markup Language + OIL*), *OWL* (*Ontology Web Language*). Ontološki riječnik nadograđuje prethodni sloj većim brojem semantičkih veza između podataka. *Ontology vocabulary* [Slika 6.] se sastoji od: *SPARQL* upita (*SPARQL query*), ontologije, pravila *RIF/SWRL* i taksonomije *RDFS*. *SPARQL* je jezik za pretraživanje ontologija, pretražuje triple-ove unutar ontologije. *SPARQL* je obrađen u narednim poglavljima. *RIF* (*Rule Interchange Format*) je *XML* jezik za izražavanje pravila koja neko računalo provodi. Razvijen je s idejom da web aplikacije mogu međusobo razmjenjivati pravila omogućujući pritom međuobno razumjevanje različito definiranih pravila na različitim sustavima. *SWRL* (*Semantic Web Rule Language)* je također jezik za izražavanje pravila napravljen kao dodatak *OWL*-u (*Ontology Web Language*) i u praksi je puno korišteniji.



Slika 6. Podjela četvrtog sloja arhitekture semantičkog weba (Ontology vocabulary)

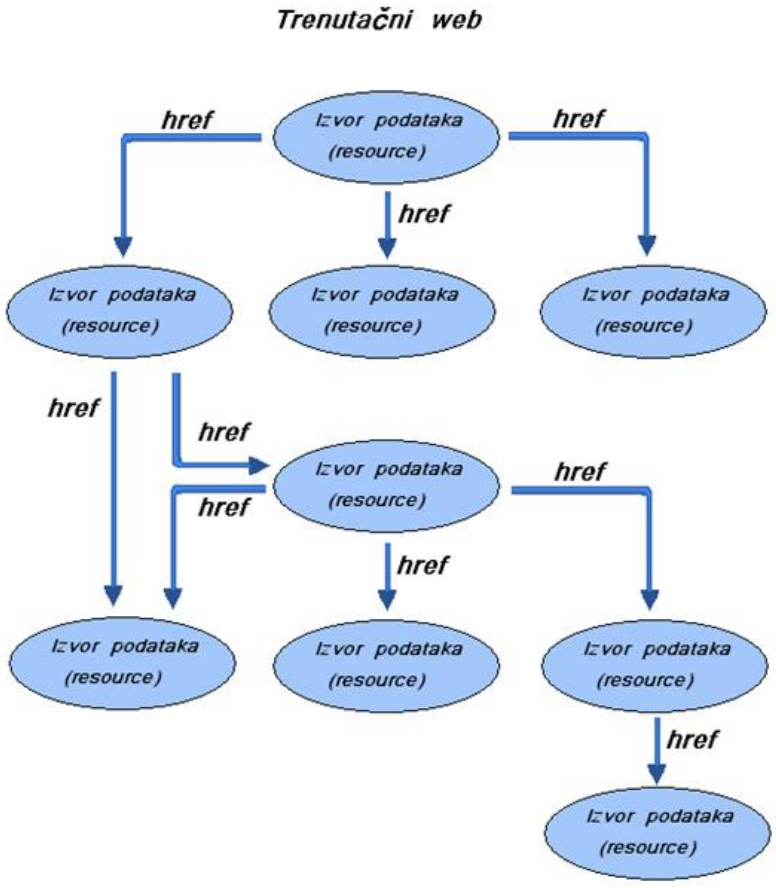
Peti, šesti i sedmi sloj arhitekture semantičkog weba zaduženi su za logiku, dokaze i vjerodostojnost. U tim slojevima vrši se razumljivo izvlačenje informacije i donošenje odluka, zaključivanje na temelju podataka i relacijama između njih (*Logic*). Nakon pronalaska podatka, utvrđuje se njegova točnost (dokaz, engl. *proof*) i provjerava se je li izvor uistinu vjerodostojan, odosno da li se može vjerovati izvoru da su podaci točni. Treći, četvrti, peti i šesti sloj spadaju pod digitalni potpis (*digital signature*). Vizija semantičke pretrage podataka je razvijanje takvog softwarea koji će unaprijediti dobivanje rezultata prosječnoj osobi tako što će prevoditi prirodne upite i vraćati semantički važne rezultate. Kao zadnji sloj arhitekture semantičkog weba, može se dodati korisničko sučelje pomoću kojeg korisnci pristupaju željenim podacima.



Slika 7. Protok informacije

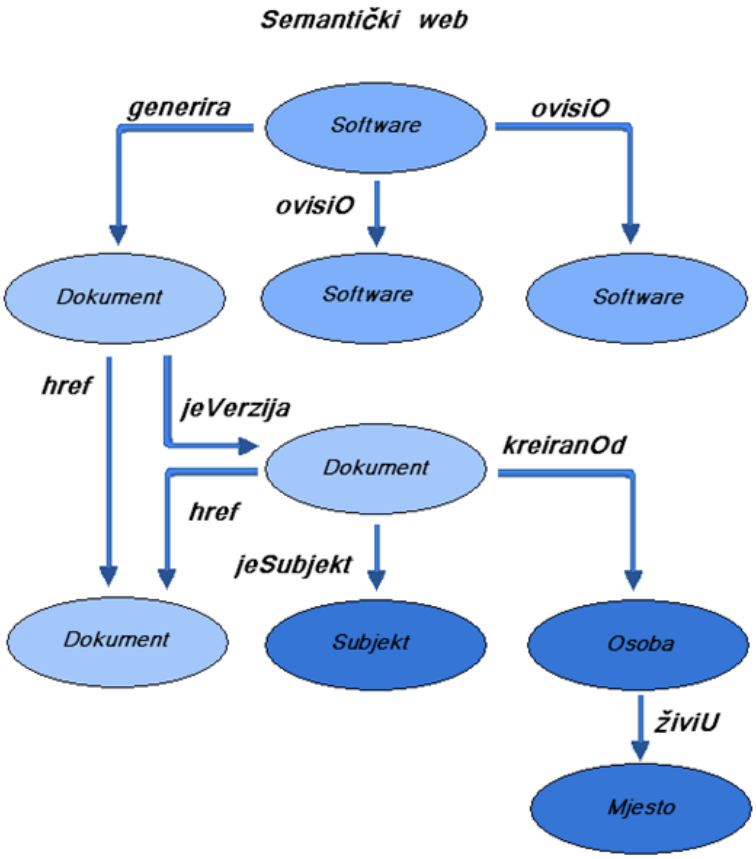
2.2. Glavni principi rada semantičkog weba

Kod izrade semantičkog weba želi se postići da se metapodaci ne odnose samo na sintaksu mrežnih resursa, nego također na njihovu semantiku, s ciljem da zamjene „mrežu poveznica“ (*web of links*) s „mrežom značenja“ (*web of meaning*). Kako je već naznačeno više puta, tradicionalno postoji značajna razlika između načina na koji je informacija predstaljena za ljudskog korisnika i način na koji je predstavljana za strojnu obradu.[16] Web je uglavnom slijedio ljudske formate, dok su strojni formati bili znatno slabije zastupljeni, što je rezultiralo, u osnovi, jednom velikom knjigom hiperveza. Semantički web teži premostiti tu veliku razliku između ljudske i strojne čitljivosti. Za semantički web postavljeni su principi koji ga obilježavaju i čine ga jednostavnim za pretraživanje informacija uz bolju standardizaciju metapodataka koji pružaju korisnicima i inteligentnim softwareskim programima pomoć u pronalaženju skrivenih podataka na webu.



Slika 8. Povezanost podataka na trenutačnom webu

Jedan od glavnih principa semantičkog weba, odnosno principa na kojima se semantički web zasniva je taj da sve može biti identificirano pomoću URI. Koristeći niz identifikatora, ljudi, mjesta i stvari u fizičkom svijetu mogu biti referirani u semantičkom webu. Onaj tko ima kontrolu na dijelu prostora imena na webu (*namespace*) može kreirati URI i postaviti ga da on identificira nešto u stvarnom svijetu.



Slika 9. Povezanost podataka na semantičkom webu

Resursi i linkovi mogu se svrstati pod neke od tipove podataka , tj. raznim linkovima i resursima mogu se dodijeliti neki tipovi podataka. Resursi su web dokumenti namijenjeni za ljudsku upotrebu i obično ne sadrže metapodatke koji opisuju čemu služe i koje su njihve relacije prema ostalim web dokumentima. Korisnik može nagađati koje veze ima resurs čitajući ga, dok je stroju puno teže doći do tih rezultata jer on to ne može shvatiti. Zato kod semantičkog weba i resursi i linkovi mogu imati tipove koji definiraju koncept. Time se daje više informacija stroju pomoću kojih on može donijeti odluke i koristiti taj podatak.

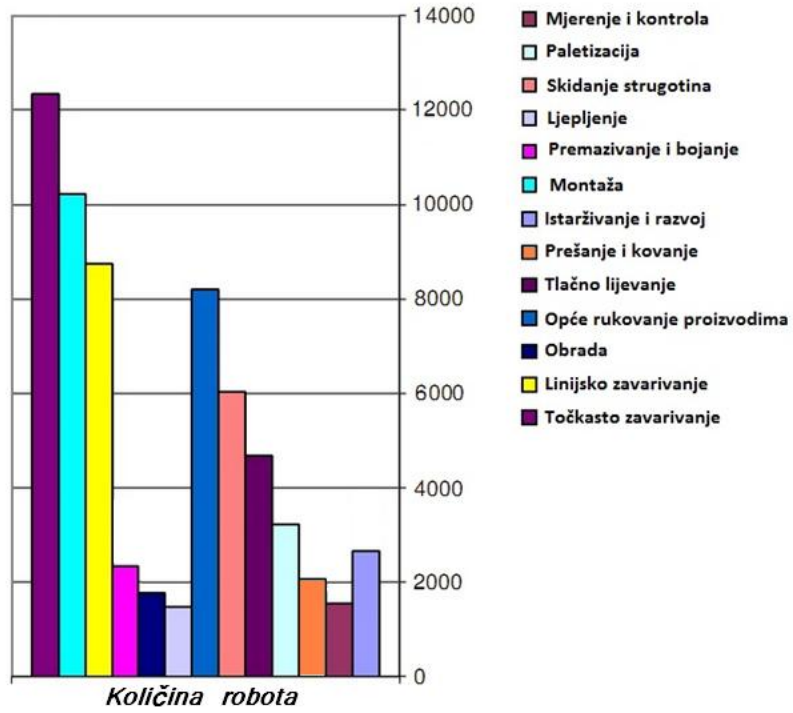
Semantički web je neograničen pa bilo tko može stvarati različite tipove ili linkove između resursa. Neki od povezanih resursa mogu prestati postojati ili adrese mogu biti ponovno iskorištene za nešto drugo. Alati na semantičkom webu trebaju tolerirati taj raspad podataka (*data decay*) i biti u mogućnosti funkcionirati unatoč njemu. Prema tome, jedna od principa semantičkog weba je tolerancija parcijalne informacije.Semantički web nema potrebe za apsolutnom istinom. Sve što je pronađeno na webu nije nužno i istinito. Semantički web ne mijenja to na bilo koji način. Vjerodostojnost ili istina se evaluira u svakoj aplikaciji koja procesira informaciju na webu. Te aplikacije odlučuju u što da vjeruju provjeravajući od kud je informacija, tj. može li se izvoru vjerovati i točnost same informacije.

Semantički web podržava evoluciju. Normalno je da su koncepti slične tematike često definirani različitim grupama ljudi na različitim mjestima na reži ili istom grupom ljudi ali u različitom vremenu. Semantički web koristi opisane konvencije koje se mogu širiti kao što se i ljudsko razumjevaje ekspandira. Konvencije dopuštaju efektivne kombinacije nezavisnih poslova raznolikih zajednica čak i kad koriste različite riječnike. Semantički web podržava evoluciju. Kod semantičkog weba u obzir se uzima i minimalistički dizajn. Cilj *W3C Activitya* je da se standardizira samo ono nužno. Implementacije jednostavnih aplikacija sad kad se baziraju na već standardiziranim tehnologijama su moguće jer semantički web čini jednostavne stvari jednostavnijima, a složene čini mogućim.

3. SUSTAV ROBOTA ZA SKLAPANJE

Robotika je znanstvena disciplina koja se zadnjih desetljeća brzo razvija, intenzitet razvoja potvrđuje i nagli porast korištenja robotskih sustava te sve veći broj znanstveno-stručnih radova. Trenutno zahtjevi za većom i fleksibilnijom proizvodnjom podrazumijevaju i veću upotrebu robotskih sustava, kako u masovnoj proizvodnji tako i u manjim proizvodnim poduzećima gdje su kvaliteta i ujednačenost proizvoda u prvom planu.

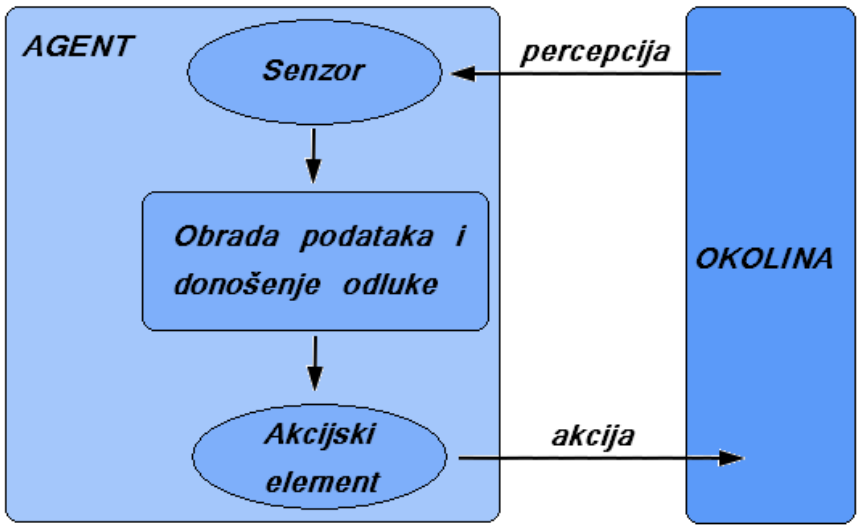
Roboti (industrijski roboti) su idealni za obavljanje teških radova te onih koji su nepogodni ili opasni za ljude. Koriste se i na mjestima proizvodnje gdje se radnje ponavljaju više puta i kao takvi se smatraju monotonim. Robotski sustavi se uspješno koriste u većini moderniziranih industrijskih grana te se može reći da su mogućnosti primjene robota u proizvodnji gotovo neograničene. Robotska tehnologija ima u osnovi svrhu nadomjestiti ljudski rad, ne samo u industriji već i u drugim djelatnostima, npr. medicini, spašavanju, istraživanju, prilikom pomaganja hendikepiranim osobama, itd. Stoga se razvoj robotike sve više usmjerava prema ostvarivanju adaptivnih, antropomatskih i spoznajnih osobina. Robot takvih osobina mora biti sposoban interpretirati i razumjeti okolinu da bi se mogao prilagođavati stalnim promjenama koje karakteriziraju realni prostor djelovanja.[6]



Slika 10. Količina robota u industriji po područjima primjene

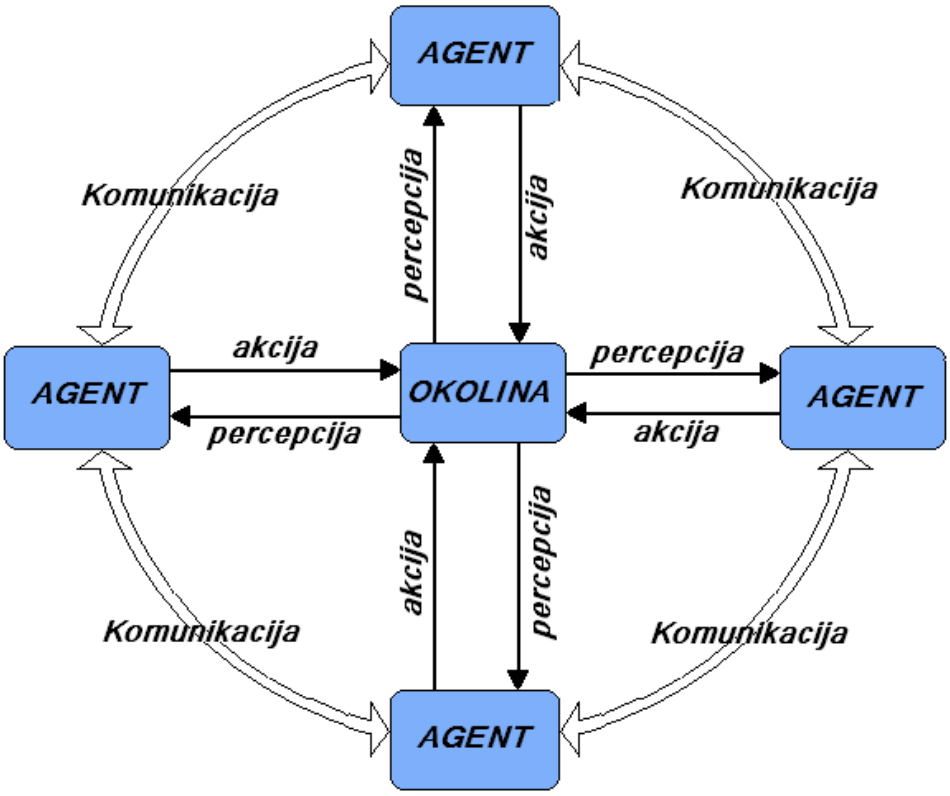
3.1. Višeagentni sustavi

Višeagentni sustavi (*M.A.S Multi-Agent System*) je računalni sustav koji se sastoji od više interaktivnih inteligentnih agenata unutar neke okoline. Višeagentni sustavi mogu se koristiti za rješavanje problema koje je teško ili nemoguće riješiti pomoću individualnog agenta ili monolitnog sustava. Agent je računarni sustav koji je sposoban samostalno izvesti neku akciju (odluku na temelju dobivenih podataka) u ime korisnika koja je u skladu s ispunjenjem zadanih ciljeva bez dodatnih uputa ili instrukcija. Svaki sustav koji se sastoji od više objekata i koji se po određenim kriterijima može rastaviti na osnovne dijelove može se smatrati višeagreatnim sustavom.[6] Tipično se pri spominjanju višeagentnih sustava misli na software agente. Međutim, agenti mogu kao u ovom slučaju biti i roboti. Agent može biti bilo koji sustav koji može opažati okolinu korištenjem nekakvih senzora, te djelovati u skladu s postavljenim ciljevima. Agenti se mogu podijeliti u tri skupine: pasivne agente (jednostavni bez ciljeva), aktivni agenti (s jednostavnim ciljevima) i kompleksni agenti (sadrže kompleksnije proračune). Okolina iz koje agenti putem senzora pribavljaju informacije također se može podijeliti u tri vrste: virtualna okolina, diskretna okolina i kontinuirana okolina. Slika [Slika 11.] prikazuje model interakcije između agenta i okoline. Prikazani model uključuje prikupljanje informacija iz okoline koje se u nekom određenom vremenu mijenjaju, pri tome agent snima trenutačno stanje.



Slika 11. Agent: model interakcije

Informacije se prikupljaju pomoću dostupnih senzora, na temelju prikupljenih informacija vrši se obrada i zaključivanje te generiranje prikladne akcije agenta u skladu s zadanim ciljem. Akcija se šalje izvršnom elementu (akcijski element) koji vrši generirane akcije u okolini. Agent pomoću subjekta zaključivanja, učenja, planiranja, prikupljanja znanja i drugih metoda umjetne inteligencije u određenoj mjeri natoji optimizirati svoje djelovanje. Navedeni subjekti u ovisnosti o ciljevia djelovanja agenta u njegovoj okolini implementirani su u funkciju modela agenta. U ovom radu cilj agenta je prilagoditi se trenutnom stanju, uz moguće dodatke agentu se može omogućiti relativno samostalno djelovanje.

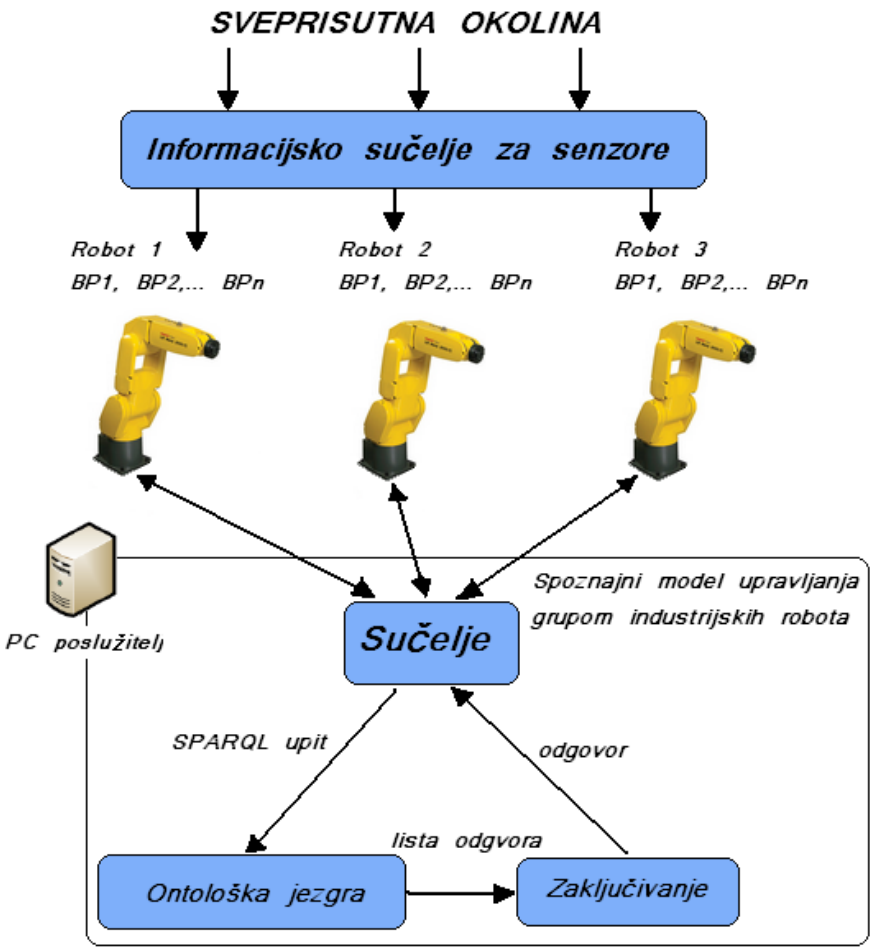


Slika 12. Višeagentni sustav: model interakcije

Kod višeagentnih sustava agenti uz to što prikupljaju podatke iz okoline, komuniciraju i međusobno. Na slici [Slika 12.] prikazan je takav višeagentski sustav s četiri agenata odgovarajućom okolinom. Donošenje odluka kod agenata unutar grupe i za višeagentski sustav kao cjelinu rješava se uzimajuću u obzir postavljene odgovarajuće kriterije i trenutno stanje okoline. U ovom radu agent koji koristi sedam senzora za prikupljanje informacija o okolini, na temelju stanja okoline sustav generira željeno ponašanje industrijskog robota.

3.2. Arhitektura robotskog sustava

Važan čimbenik prilikom razvoja autonomnih sustava je prilagodba sustava stalnim promjenama prostora. Svaki proces, objekt, prostor jedinstven je po svojoj prirodi. Ukoliko je agent sposoban koristeći znanje, percepciju ili inteligenciju donijeti odluku o svojem budućem djelovanju koje nije u potpunosti i u svakom trenutku jasno definirano možemo zaključiti da u određenoj mjeri može kontrolirati svoju okolinu. Da bi bio u mogućnosti to napaviti agent mora posjedovati određeno znanje, uključujući procese i ostale relevantne komponente. Zato razvijeni robotski sustav sadrži senzore integrirane u okolinu po principima sveprisutnog računarstva u svrhu sakupljanja informacija. Robotski sustav također ima razrađen semantički opis domene djelovanja koja uključuje opis okoline zajedno sa svim relevantnim procesima i objektima. Senzori iz okoline osiguravaju neprekidan dotok informacija koje se potom dovode na ulaz u ontološku jezgru modela.



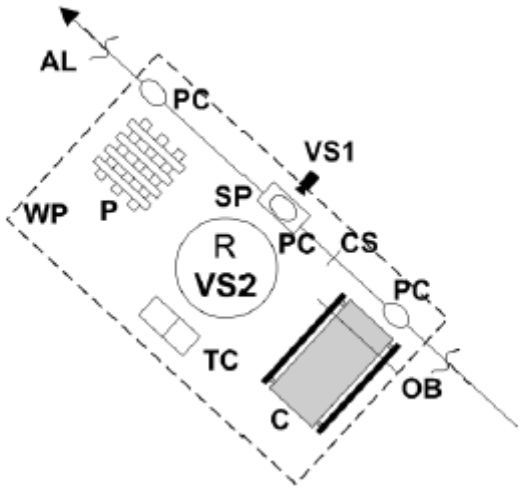
Slika 13. Arhitektura sustava

U robotskoj montaži informacije o okolini, poziciji i orijentaciji elemenata u procesu predstavljaju esencijalne podatke za razvoj i primjenu kontrolnih mehanizama. Slika [Slika 13.] prikazuje je arhitekturu sustava. Sva potrebna software i hardware podrška (sučelje i ontološka jezgra zajedno sa zaključivanjem) nalazi se na lokalnom osobnom računalu (*local PC*). Ontološka jezgra rađena je u *TopBraid Composer* alatu koji u sebi ima integriranu aplikaciju za isprobavanje *SPARQL* upita na kojem je vršena provjera *SPARQL* koda. Nakon što se *SPARQL* upitom dobije lista s rješenjima, sustav za zaključivanje (Bayesova mreža) će na temelju procjene robotu vratiti jedinstveno rješenje u vidu jednog uzorka (obrazca) ponašanja. Unutar stvarnog postava, roboti će s poslužiteljskim računalom na kojem je spoznajni model pohranjen komunicirati pomoću *Socket Messaging* poruka koje se temelje na *TCP/IP* mrežnoj komunikacijskoj tehnologiji. Rad dostavne trake bit će upravljan posebnim kontrolerom koji je povezan s robotima pomoću *DeviceNet* mrežne komunikacijske tehnologije. Preko toga kontrolera roboti će moći po potrebu uključivati i isključivati signale koji će propuštati ili zaustavljati nosače proizvoda na dostavnoj traci unutar promatranog radnog mjesta (*BP1*, *BP2*).

Slika 14. Robotski Sustav Laboratorija za projektiranje izradbenih i montažnih sustava

Robotski sustav koji s koristi u ovom radu sastoji se od tri (broj robota u sustavu se može mijenjati) šest-osna robota smještena uz montažnu liniju. Svaki od robota sadrži pripadajuće senzore i ostalu opremu i predstavlja zasebnu montažnu grupu koja je semantički opisana unutar razvijenog spoznajnog modela. Robotski sustav se nalazi u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava [Slika 14], Fakulteta strojarstva i brodogradnje, Sveučilišta u Zagrebu, i koristi se za različite namjene. Sustav koristi više vrsta senzora smještenih u okoliš u svrhu pravovremenog i primjerenog reagiranja na promjene, zaustavna mjesta koja omogućuju precizno sklapanje, konvejere za unos dijelova u sustav te palete za pohranu proizvoda.[6]

Zbog lakše izrade ontologije, okoliš robotskg sustava podijeljen je u dva dijela: liniju za sklapanje (*Assembly Line*) i radna mjesta (*Working Place*). Linija za sklapanje sadrži nosače dijelova (*Part Carriers*), te može sadržavati i dodatne senzore u ovisnosti o potrebama aplikacije. Jedno radno mjesto [Slika 15.] sadrži konvejer (*C-Conveyor*), paletu (*P-Pallet*), vizijski senzor smješten iznad zaustavnog mjesta (*VS1 –Vision Sensor 1*), robota (*R-Robot*), vizijski senzor smješten na robotu (*VS1–Vision Sensor 2*), izmjenjivač alata (*TC–Tool Changer*), te zaustavno mjesto (*SP–Stopping Place*) i kapacitivni senzor (*CS–Capacitive Sensor*) smješteni na liniji. Radna mjesta su međusobno povezana linijom za sklapanje koja omogućuje transport nosača dijelova. Nosači dijelova se koriste prilikom transporta ugradbenih elemenata (dijelova), sklopova ili gotovih proizvoda. Komponente radnog mjesta čine senzori (na zaustavnom mjestu, kapacitivni senzor, izmjenjivač alata, itd.), robot - agent, konvejer i paleta.[6]

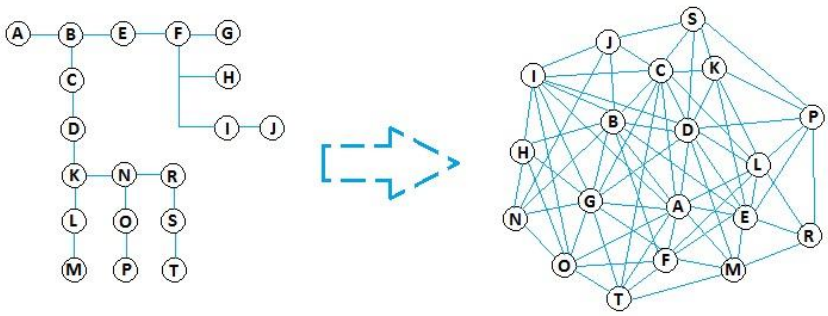


Slika 15. Izgled radnog mjesta

4. ONTOLOGIJA

Ontologija (grč. *ontos* = biće, stvarnost; grč. *logos* = riječ, učenje) u svom izvornom filozofskom značenju, predstavlja nauku o biću, o onome što postoji, učenje o općim, fundamentalnim i konstruktivnim određenjima bitka.

U tehničkom smislu, ontologija nekog sustava predstavlja točno definirane pojmove ili koncepte i njihove međusobne odnose. Ontologija se koristi kod semantičkog weba gdje je iskorištena za jednostavniji i poboljšani način pretraživanja. Zbog točnosti ontologija pretraživanje weba postaje preciznije, jer se ne pretražuju samo pojmovi već sve ostale informacije vezane uz traženi koncept. Ontologije omogućuju grupiranje, točno definiranje i međusobni odnos podataka. Kod ontologije je bitno da je ona što detaljnije opisana, tj. domena koja se opisuje da je što bolje definirana. U nekim slučajevima predetaljan opis može biti i problematičan jer u tom slučaju ontologija postaje prekompleksna. Način izrade ontologije treba biti takav da je ontologija što jednostavnija, a da sadrži sve potrebne informacije i da su odnosi točno definirani. U suprotnom, ontologiju je teško održavati i proširivati.



Slika 16. Svojstvo ontologije u odnosu na konvencionalne hijerarhijske strukture

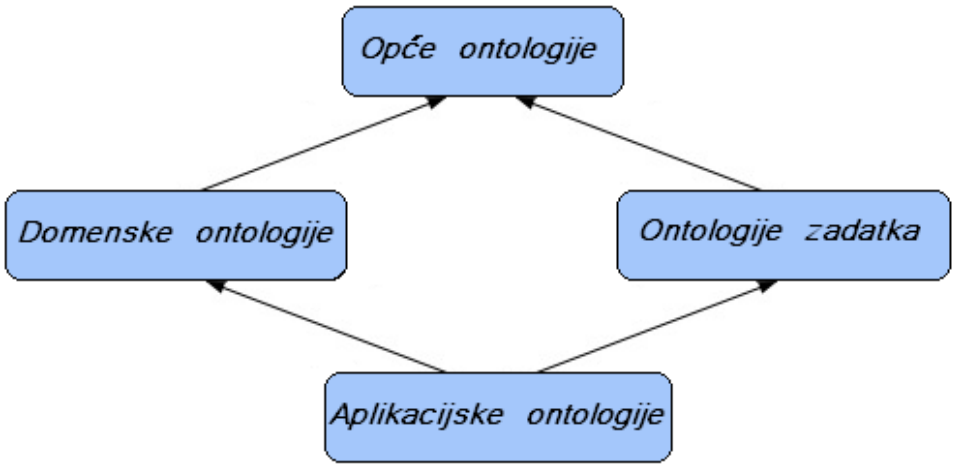
Ontologije mogu biti složene (engl. *heavyweight*) i jednostavne (engl. *lightweight*) i mogu se konstruirati različitim tehnikama modeliranja i biti implementirane u različitim jezicima. Ontologije mogu prezentirati visoku razinu informacija (engl. *highly informal*), ako su zapisane prirodnim jezikm, mogu biti ograničeno informativne (engl. *semi-informal*), *ako* se zapisuju u restriktivnoj i strukturiranoj formi prirodnog jezika, te poluformalne (engl. *semi-formal*), ako su zapisane u umjetnoj i formalno definiranom jeziku poput OWL-a (*Ontology Web Language*)[18].

Kao što je već navedeno, ontologija treba biti čitljiva i čovjeku i računalu, što kod ontologija koje su zapisane u prirodnom jeziku nije slučaj. Iz tog razloga potrebno je poslužiti se tehnikama koje se upotrebljavaju u području programskog inženjerstva kako bi se izgradile veze između koncepata i pojednostavnila ontologija. Ontologija kao opis koncepata i veza unutar hijerarhije nekog sustava predstavlja najvažniju komponentu semantičkog weba.

Zbog važnosti ontologija u semantičkom webu one se također mogu smatrati jednim od važnijih komponenti u informativnim poslovnim sustavima. Ontologija nekog sustava se sastoji od koncepata (klasa, objekt/instanca) i relacije između njih. Relacije mogu biti: hijerarhije klasa (podklasa, superklasa, sestrinska klasa), svojstva (*properties*), ograničenja vrijednosti (*allValuesFrom*, *someValuesFrom*), isključivi izrazi (*disjoint*) i specifikacije logičkih relacija između instanci. Izradom ontologije pomoću navedenih relacija dobiva se baza znanja koju je moguće jednostavno iznova upotrebljivati i dijeliti iz određenih domena. Postoje razne klasifikacije tipova ontologija, jedna od jednostavnijih klasifikacija je prema stupnju njihove ovisnosti o zadatku [Slika 17.][19].

Klasifikacija prema zadatku:

- opće ontologija
- domenske ontologije
- ontologije zadatka
- aplikacijske ontologije



Slika 17. Klasifikacija ontologija (Guarino, N.)

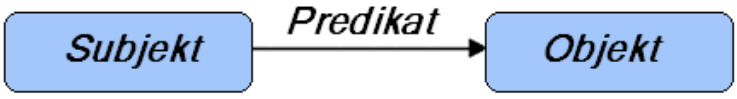
Ontologija nudi fleksibilne ulazne točke, jer se svaka specifična perspektiva u ontologiji može pratiti i dovesti u vezu sa svim njegovim povezanim konceptima. Svaka ontologija sadrži koherentnu navigaciju tako što omogućava kretanje od koncepta do koncepta u strukturi ontologije. Veze koje povezuju koncepte su vidljive a one koje ističu bitne informacije dostupne su bez prethonog znanja o domeni ili njezinoj terminologiji.

4.1. Ontološki jezici

RDF, RDFS i *OWL* su osnovni ontološki jezici za prikaz semantičkog weba, gdje je *RDF* osnova. *RDF* rješava osnovni problem pri izradi semantičkog weba: upravljanje uređenim podacima. Svi ostali standardi vezani uz semantički web nastali su temelju tako uređenih podataka. Web koji poznajemo je sastavljen od niza dokumenata koji su povezani jedan s drugim. Vezu između stvarnog svijeta i teksta koji piše u dokumentu može stvoriti jedino čitatelj tog dokumenta. Kod semantičkog weba stvari iz stvarnoga svijeta nazivamo izvorima (*resource)* koji mogu biti bilo što, ovisno što nas zanima. U mreži informacija svatko može doprinijeti ukupnom znanju o našem izvoru.

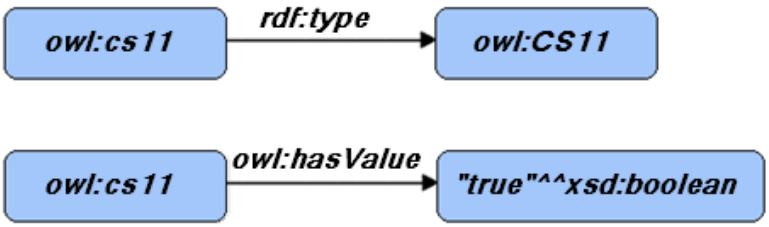
4.1.1. RDF (Resource Description Framework)

RDF (Resource Description Framework – sustav za opisivanje izvora) je prvo i osnovno sustav za modeliranje (oblikovanje) podataka. Pomoću *RDF*-a svaka veza između dva podatka elemenata je eksplicitno prezentirana čime se omogućuje izrada veoma jednostavnih modela. Veze pomoću *RDF*-a se prezentiraju u formi: subjekt – predikat – objekt (*subject –predicate – object)*. Pri izradi modela pomoću *RDF*-a povezivanje podataka je veoma jednostavno jer se svi podaci iz svih izvora povezuju na jednom mjestu (ne izrađuju se posebni stupci kodova već se sve piše u jednom prozoru u kojme se onda svi podaci povezuju).



Slika 18. Općeniti oblik RDF trojke

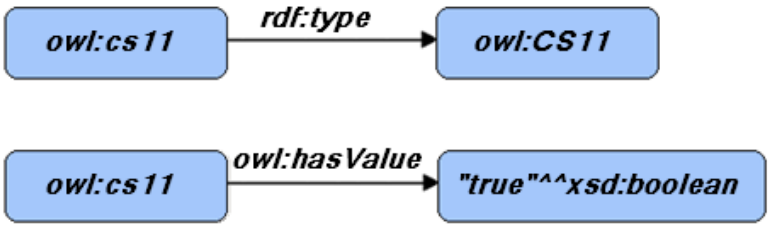
RDF se ispisuje u raznim tekstualnim oblicima. Najjednostavniji oblik pisanja *RDF*-a se zove *N-Triples* i najsličniji je izvornom pisanju *RDF*-u. Odnosi se na pisanju izvora koristeći potpune nekraćene *URI*-e (*Uniform Resource Identifier).* Svaki identifikator *(URI)* se piše unutar izlomljenih zagrada „<“ i „>“. Tri izvora se postavljaju u trojku subjekt – predikat –objekt nakon čega slijedi točka (.). *N-Triples* kod je podosta dugačak te se rijetko koristi pri direktnom upisivanju. Na slici [Slika 19.] vidljiv je jedan od primjera *N-Triples*-a. Kod je predugačak da bi stao na stranicu A4 formata je je izlomljen u dva dijela, lom je prikazan oznakom #. Kod prikazuje da je instanca *owl:cs11* tip (*rdf:type*, skraćeno se može pisati i samo „*a*“) klase *owl:CS11,* i da ista ta instanca *owl:cs11* ima vrijednost (*owl:hasValue) true* (dodataka *xsd:boolean* označava da je vrijednost u navodnicima binarne vrste, može biti samo istina ili laž- *true* ili *false*).




```
<http://www.w3.org/2002/07/owl#cs11> #
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#CS11> .
<http://www.w3.org/2002/07/owl#cs11> #
<http://www.w3.org/2002/07/owl#hasValue> "true"^^<http://www.w3.org/2001/XMLSchema#boolean> .
```

Slika 19. Primjer N-Triples RDF-a

Kompaktnija verzija pisanja *RDF*-a i češće korištena zove se *Turtle* [Slika 20.]. *Turtle* je kombinacija prikaza trojki iz *N-triple* oblika i sažetim prikazom *qname (qualified name)*. *Qname* je kompaktnija verzija *URI*-a. Kod zapisivanja *RDF*-a *N-Triple* oblikom često zbog ponavljanja prvog dijela *URI*-a dolazi do nepreglednosti. Kako bi se izbjeglo ponavljanje i doprinijelo preglednosti zapisa u datoteke se uvodi *NameSpace* (*NS*). Tako za primjer *URI*-a: *http://www.sjever.fsb.hr/robotic_assembly,* kako ne bi morali svaki puta ponavljati ovaj podatak, možemo ga podijeliti u dva dijela. Drugi dio je uvijek različit i trenutno nebitan. Prvi dio je domena *URI*, odnosono *NS* koji ćemo zajeniti s nekim prefiksom (npr. prefiks *fsb*). Prefiks je definiran: *xmlns:fsb= http://www.sjever.fsb.hr/.*




```
owl:cs11
      a       owl:CS11 ;
      owl:hasValue "true"^^xsd:boolean .
```

Slika 20. Primjer Turtle RDF-a

Dok je *Turtle* verzija kompaktnija za ljude, mnoge web infrastrukture su prikazuju informaciju pomoću *HTML*-a, obično *XML*-om (*EXtensible Markup Language)*. *W3C* je stoga predstavio *XML* serializaciju *RDF*-a nazvanu *RDF/XML*. Informacija koju sadrži *Turtle* kod sadržana i u *RDF/XML* kodu uz dodanu deklaraciju *qname*-ova. *RDF/XML* koristi niz pravila kojima se određuje ispisivanje potpunog *URI*-a nekog dokumenta.

4.1.2. RDFS (RDF Schema)

RDFS je jezična shema za *RDF*, opisuje veze za razne tipove objekata (*Classes*), hijerarhijske odnose između njih (*subClasses*), svojstva koji opisuju objekt (*Properties*), i odnose između njih (*subProperty*). *RDFS* dopušta i dodavanje raspona i domene kojima su neka svojstva ograničena i pripisivanje svojstva pojedinačnim primjerima (svojstva koja su specifična za neki odabrani primjer, niti jedan drugi primjer ne može imati ta svojstva). Navedene značajke čine *RDFS* fragmentom deskriptivnih logika u smislu izražajnosti.

Sustav klasa u *RDFS*-u uključuje jednostavni princip nasljeđivanja na temelju uključenih podataka. [20] Ukoliko je jedna klasa podklasa druge znači da je instanca jedne klase ujedno i instanca druge klase. *RDFS* jezik se ispisuje pomoću *RDF*-a, koji je osnova ostalim ontološkim jezicima. Prema tome, sve shematske informacije (klase, podklase, svojstva, podsvojstva, domena, raspon) prikazuju se pomoću *RDF* trojki. *RDFS* je još uvijek logički prejednostavan da izrazi velik dio onoga što je rečeno na webu. Unutar njegovog okvira, moguće je deklarirati nove klase i naseliti ih primjerima, ali se ne može reći ništa dodatno o tim klasama i primjerima. Ne mogu se postaviti aksiomi ili pravila zaključivanja koja se na njih odnose [16].

RDFS nazivi su indeksirani pomoću naziva čija detaljna značenja nisu poznata. Problem definiranja značenja na strojno razumljivi način koji bi se riješio u jednom ontološkom jeziku ostaje neriješen.

4.1.3. OWL (Ontology Web Language)

OWL je jezik za opisivanje ontologija kojeg karakterizira veći stupanj složenosti od *RDF*-a i *RDFS*-a. Kao i *RDFS*, *OWL* se također zasniva na sintaksi *RDF*-a. Nastao je kao potreba za proširenjem *RDFS*-a a izveden je pomoću *DAML* i *OIL* ontoloških jezika. *OWL* se također može prikazati u vidu grafa pomoću *RDF* trojki.

Skupovi podataka koji se opisuju *OWL* u jednom svom pogledu već predstavaljaju neko znanje. Može se reći da je *OWL* zbog mogućnosti izvođenja činjenica koje nisu eksplicitno navedene postao jezik za predstavljanje znanja. *OWL* je moguće preciznije opisati osobine informacijskih sustava. Ontologije napisane u *OWL*-u mogu se jednostavno kombinirati s već napisanim ontologijama (također u *OWL*-u) što je ujedno i omogućilo brz razvoj *OWL* jezika.

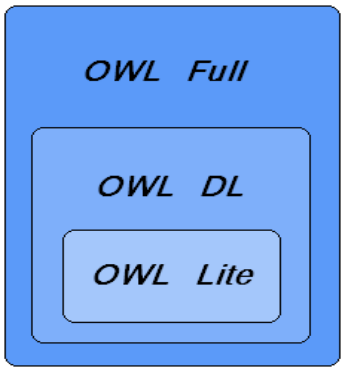
Okvir *OWL*-a uz postojeće mogućnosti *RDF*-a i *RDFS*-a pridodaje mogućnost definiranja kardinalnosti (minimalnu, maksimalnu ili točno određenu vrijednost) nekog svojstva u odnosu na neku klasu. Tako se može definirati klasa koja npr. ima točno određen broj svojstava i svaka instanaca koja sadrži taj točan broj svojstava pripada toj klasi, a bilo koja druga (koja ima više ili manje svojstava) ne može pripadati toj klasi.

OWL-om se također mogu definirati sinonimi za objekte, klase i svojstva za koja se potom mogu definirati karakteristike: tranzitivnost, simetričnost, inverzno svojstvo. Jedno od važnijih svojstva *OWL* je i kreiranje novih klasa primjenom skupovnih operacija nad postojećim klasama.

Kod *OWL* definirana su tri različita podjezika:

- *OWL Lite*
- *OWL DL*
- *OWL Full*

OWL podjezici nastali su zbog kompromisa koji je napravljen između izražajnosti jezika i efikasne podrške za logiku zaključivanja.



Slika 21. Podjela OWL-a

4.1.3.1. Owl Lite

OWL Lite je najjednostavnija verzija *OWL*-a. Sadrži brojna ograničenja koja ostala dva podjezika ne sadrže. Karakterizira ga jako dostatno ograničenje sintakse zbog pojednostavljenja sintakse. Upravo to predstavlja razlog upotrebe u aplikacijama gdje je dovoljna gotovo minimalna ekspresivnost ontologija. *OWL Lite* predstavlja dobru polaznu osnovu za izradu ontoloških alata. Zbog svoje jednostavnosti namjenjen je uglavnom za podržavanje jednostavnih ograničenja i hijerarhijske klasifikacije. *OWL Lite* ne sadrži neke od često korištenih jezičnih konstrukcija, stoga je potrebno pripaziti pri odabiru *OWL* podjezika. *OWL* ne sadrži sljedeće češće korištene jezične konstrukcije: *disjointWith, equivalentClass unionOf, intersectionOf, complementOf, hasValue, minCardinality, maxCardinality, cardinality.*

4.1.3.2. OWL DL (OWL Descriptive Logic)

OWL DL (*OWL Descriptive Logic*) je podjezik *OWL*-a s određenim ograničenjima, puno manje nego *OWL Lite.* Može se reći da je *OWL DL* proširenje *OWL Lite* podjezika. Ima uvedeno manje ograničenja tako da je izražajno bolji od podjezika *OWL Lite,* ali lošiji od podjezika *OWL Full*. Najznačajnije ograničenje je da klasa ne može biti instanca ili svojstvo i da svojstvo ne može biti instanca ili klasa.

OWL DL ima dobru formalnu osnovu jer se zasniva na deskriptivnoj logici, od kud je i dobio ime (*DL-Descriptive Logic*). Prednost ovog podjezika je u tome što dozvoljava efikasnu podršku za zaključivanje, osiguran stupanj složenosti računskih operacija, te visoka odlučnost koja jamči da će sve operacije biti izračunate, i to u konačnom vremenu.

4.1.3.3. OWL Full

OWL Full je najizraženiji od navedena tri podjezika *OWL*-a. Karakterizira ga maksimalna izražajnost te je sintaksno neovisan o *RDF*-u. *OWL Full* ne sadrži ograničenja. Sadrži, kako i samo ime kaže (*Full*), kompletan *OWL* jezik. Osnovna karakteristika jezika je da jedna klasa, koja je po definiciji skup instanci, može i sama po sebi biti instanca. Zbog navedene karakteristike ovog podjezika vrijeme potrebno za obradu na njemu zasnivanih modela je dugotrajno. Zbog toga se koristi samo tamo gdje je iznimno potrebno. Zbog svojih karakteristika može se reći da je *OWL Full* proširenje *OWL DL*-a koji predstavlja proširenje *OWL Litea*. Svaka ontologija nastala u *OWL Lite*a ujedno je i *OWL DL* i *OWL Full* ontologija, dok je *OWL DL* ontologija ujedno i *OWL Full* ontologija.

4.2. Alat za izradu ontologija TopBraid Composer

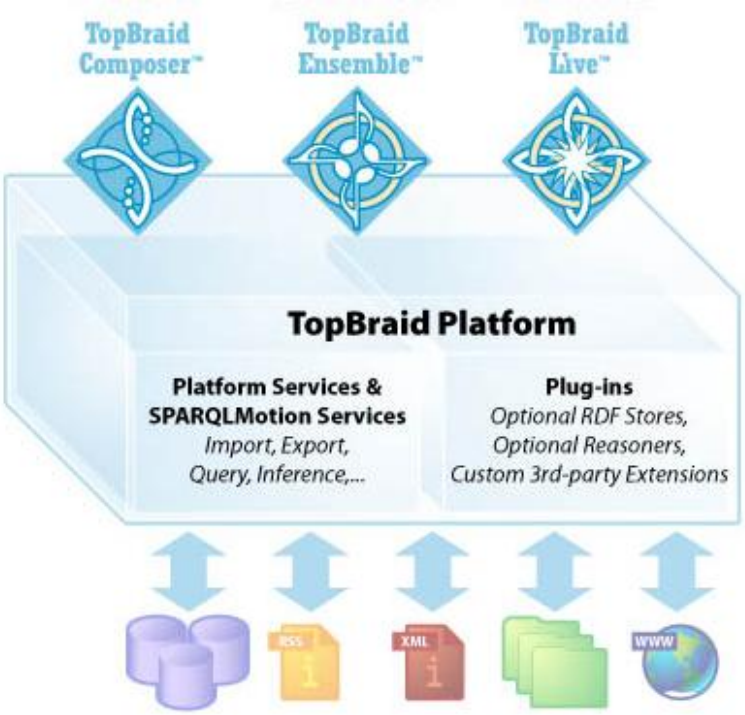
TopBraid Suite je paket alata za izradu integriranih aplikacija koje se uklapaju u postojeću okolinu informacijskih tehnologija. Sve komponente *TopBraid* paketa funkcioniraju unutar platforme s otvorenom arhitekturom izgrađenom posebno za implementaciju W3C standarda za integriranje i povezivanje podataka iz različitih izvora.

Alat unutar *TopBraid* paketa koji se koristi za povezivanje podataka, dizajniranje *query*ja (upita za pretraživanje), modeliranje pravila te ukupne ontologije zove se *TopBraid Composer*. *TopBraid Composer* je jedan od najboljih alata za unos ontologije i *IDE* (*Integrated Development Environment*) za izradu semantičkih aplikacija.

Za izradu potrebne ontologije koristi se *Maestro Edition* (*TBC – ME*) verzija koja je optimizirana za izradu web aplikacija i usluga baziranim na *TopBraid Live* platformi. *TopBraid Live* je drugi alat koji služi za postavljenje napravljenih aplikacija s ontologijama na server. Alat *TopBraid* paketa koja služi za izradu aplikacija za korištenje i pretraživanje ontologija naziva se *TopBraid Ensemble*.

Prema tome, *TopBraid* paket alata se dijeli na:

- *TopBraid Composer* (služi za dizajniranje)
- *TopBraid Ensemble* (služi za sastavljanje aplikacija)
- *TopBraid Live* (služi za interakciju)



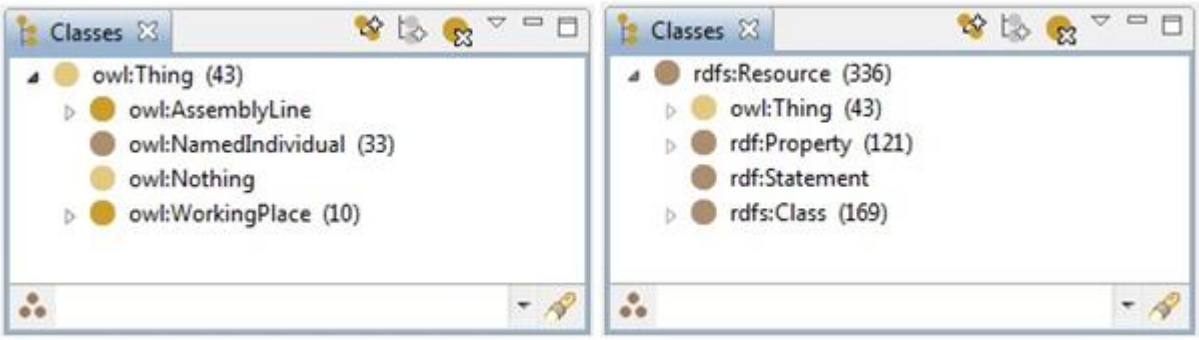
Slika 22. TopBraid paket alata

TopBraid Composer Maestro Edition je dostupan u probnoj verziji (engl. *trail version*) u trajanju od 30 dana. *TBC – ME* sadrži i vlastiti interni web server za testiranje aplikacije te sadrži alate za razvoj aplikacija: *SPARQL Rules*, *SPARQL Web Pages* i *SPARQLMotion data processing*. Zasnovan je na *Eclipse* platformi te omogućava potpunu podršku formatima *owl* i *rdfs*, te sadrži *inference engine*, odnosno funkciju za izvođenje zaključaka i mogućnost izvršavanja *SPARQL* upita.

Alat za izradu aplikacija *TopBraid Composer* sastoji se od niza prozora koji služe za prikaz i uređivanje podataka i ontologije. Jedan od najbitnijih je prozor s prikazom hijerarhije (stablo, engl. *tree*) klasa s nazivom *Classes*. Svaka hijerarhija (ontologijsko stablo) započinje s jednom super-klasom pod nazivom *owl:Thing*. Klasa *owl:Thing* je preddefinirana klasa i sve druge klase su njezine podklase.

Klasa je skup objekata, odgovara konceptu deskriptivne logike (*description logic*). Klasa može imati članove (članovi mogu biti ili instance ili druge klase). Član može pripadati nijednoj, jednoj ili više klasa. Klase u pravilu započinju prefiksom (engl. *prefix*, npr. *owl:MojaKlasa*, gdje oznaka *owl* predstavlja prefiks, ali je moguće i ostaviti prefiks prazan, takve klase izgledaju slično klasi → *:MojaKlasa*), te velikim početnim slovom npr. *Klasa*. Ukoliko se ime klase sastoji od više riječi, one se pišu zajedno s velikim početnim slovima.

Recimo da želimo olikovati neku klasu s nazivom *Assembly line* i prefiksom *owl*, ta klasa bi u hijerarhijskom stablu (ali i u cijlokupnoj ontologiji) izgledala: *owl:AssemblyLine*. Prozor s prikazom stabla klasa ima nekoliko funkcija: za izradu podklase, za izradu sestrinske klase, za brisanje klase, za promjenu prikaza hijerarhije unutar prozora, tražilicu klasa unutar hijerarhije te za minimizaciju i maksimizaciju veličine prozora.



Slika 23. Hijerarhijska stabla s različitim načinom prikaza klasa

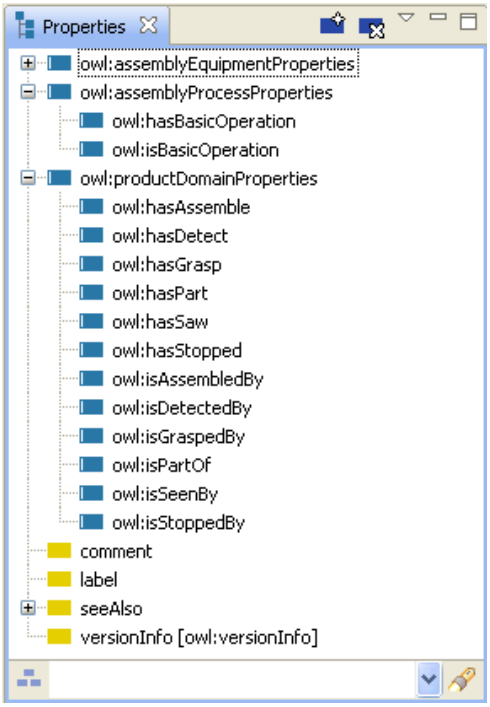
Način na koji funkcije za dodavanje klasa funkcioniraju je da se prvo odabere klasa kojoj se dalje po želji može pridodati podklasa ili sestrinska klasa željenog imena. Nakon što se željena klasa kreira, ona je automatski u ontologiji definirana kao podklasa prijašnje odabrane klase (npr. *rdfs:subClassOf → owl:Thing*) te joj je dodijeljen tip podatka, odnosno klasifikacija koja označava da je podatak koji smo unijeli klasa (*rdf:type → owl:Class*).

Funkcija za brisanje klase djeluje na istom principu, prvo se odabire klasa koja se želi obrisati te klikom na funkciju za brisanje ta klasa obriše. Valja napomenuti da ukoliko neka klasa ima podklase ili neke druge članove, njezinim brisanjem se brišu i svi podaci koji su članovi te klase (ukoliko određeni član nije član i neke druge klase). Funkcija tražilice po stablu klasa je veoma jednostavno, upisivanjem traženog naziva klase dobivamo rezultate u obliku prikaza klase u stablu (tražena klasa je označena plavim pravokutnikom), a moguće je dobiti i više odgovora za traženi pojam koji su onda prikazani u posebnom prozoru pod nazivom *Multiple matches*.

Funkcija za prikaz hijerarhije stabla nam omogućava da kao startnu klasu hijerarhije stabla prikažemo *owl:Thing* ili *rdfs:Resource* [Slika 23.]. Često je poželjnije zbog snalaženja u ontologiji postaviti kao startnu klasu *owl:Thing*.

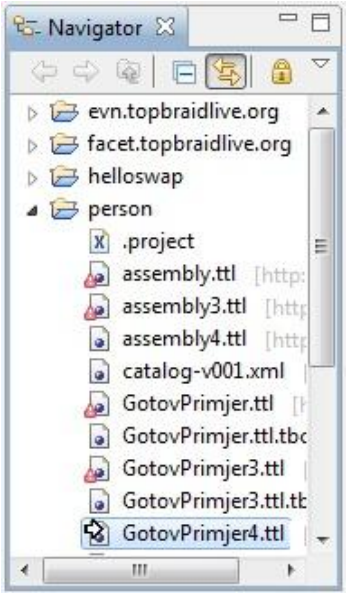
U hijerarhijskom stablu pri kreiranju nove ontologije uz preddefiniranu klasu *owl:Thing* nalazi se i preddefinirana klasa *owl:Nothing* koja nema članova (prazna klasa) i isto kao što je *owl:Thing* superklasa svim klasama tako je *owl:Nothing* podklasa svim klasama. Klasa *owl:Nothing* se ne pojavljuje u pretraživanjima kao podklasa svake klase (što je u stvari i nepotrebno) jer se taj podatak pomoću unaprijed definiranih filtera izbacuje.

Prozor *Properties* je vema sličan prozoru *Classes* te predstavlja svojstava različitih klasa kojima su povezana s drugim članovima. Prikaz svojstava je također u hijerarhijskom obliku gdje svako svojstvo može imati podsvojstvo. Svojstva su naslijediva iz klasu u podklasu, što znači da će neka klasa imati obilježja, odnosno svojstva svoje superklase. Prozor sa svojstvima ima više funkcija za izradu različitih novih svojstava, za brisanje svojstva te isto kao i prozor za prikaz klasa, ima traženje pojmova i funkcije za minimizaciju i maksimizaciju prozora. Postoje različiti tipovi svojstava koje klasa može imati. Ne postoji ograničenje broja svojstava koje neka klasa može imati niti ograničenje broja klasa koje se pomoću nekog svojstva mogu povezati s odabranom klasom. Ne postoji funkcija kojom se može stvoriti podsvojstvo (*rdfs:subPropertyOf*) nekog svojstva, već se definira neko novo svojstvo koje se zatim jednostavno mišem prenosi na prethodno definirano svojstvo. Postoje različiti tipovi svojstava te pri kreiranju novog svojstva potrebno je odabrati određeni tip.



Slika 24. Kreirana svojstva

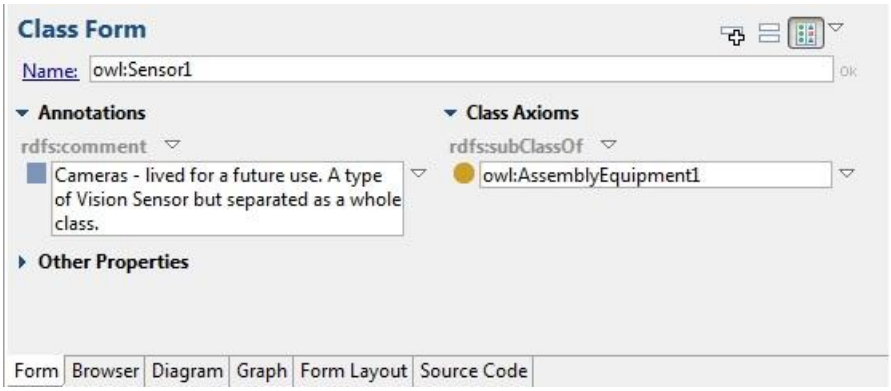
Navigator je prozor za navigaciju datotekama, dakle prikazuje nam podatke vezane uz *TopBraid Composer* datoteku *Workspace* gdje su spremljene ontologije (naše novonastale ontologije i *TopBraid* primjeri za upoznavanje aplikacija). Za pokretanje ontologije potrebno je samo dvostruko kliknuti mišem na bilo koju *OWL* ili *RDF* datoteku prikazanu u pregledniku *Navigatora*. Ontologija koja se trenutno koristi je označena malom bijelom strelicom. Ako neka ontologija u sebi ima implementirane neke druge ontologije koje koristi te ontologije su označene žutom strelicom. Ukoliko postoji više datoteka koje koriste isti *namespace* pojavljuje nam se crveni trokutić s uskličnikom. To se događa u slučaju kada istu datoteku spremimo s više različitih imena, tada *Composer* povezuje *namespace* sa zadnje obrađenom (sačuvanom) verzijom datoteke a ostalim datotekama dodjeljuje uskličnik. Datoteke se također mogu prebacivati iz jednog *foldera* u drugi „*drag&drop*" principom. Prozor *Navigator* ima funkciju zaključavanja određene datoteke te time onemogućuje daljnju izmjenu dotične datoteke.



Slika 25. Prozor Navigator

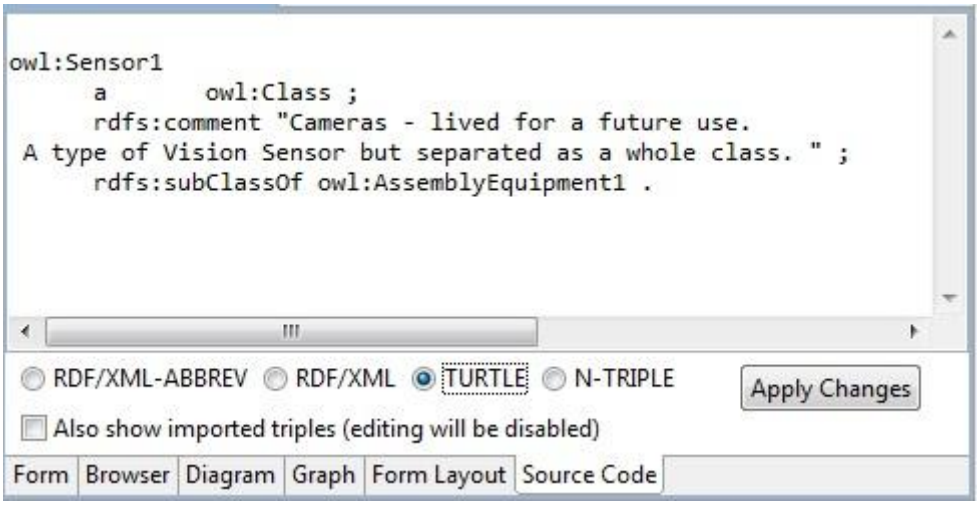
Form panel je glavni prozor, preglednik za prikazivanje i uređivanje ontologije. *Form panel* ima nekoliko načina prikaza podataka: *Form* (koji može biti *Class form* , *Object Property form* ili može imati neki drugi naziv ovisno o odabranom tipu podataka), *Source Code*, *Diagram*, *Graph*, *From Layout* i *Browser*. Najzanimljiviji su *Class form*, *Source Code* i *Browser* u kojem nije moguća izmjena podataka, ali je uređen prikaz podataka na način kakav bi bio ukoliko bi ontologiju otvorili u nekom od interenet preglednika.

Form preglednik sadrži razne sekcije za upisivanje vrijednosti s kojima je odabrani objekt povezan. Sekcije za upisivanje vrijednosti se mogu dodavati i brisati pomoću trokutića na kraju sekcije koji ima opcije za dodavanje ili brisanje odabranog sadržaja. Pregled sekcija je moguće uređivati pomoću dostupnih funkcija u desnom vrhu prozora.



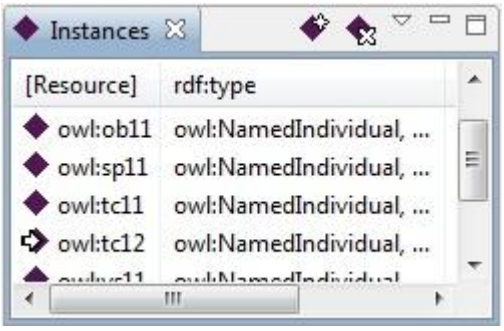
Slika 26. Form panel – form preglednik

Source Code preglednik nam prikazuje ontologiju u izvornom kodu u nekoliko osnovnih formata (*RDF/XML, turtle, N-triple*). Unutar *Source code* prikaza moguće je mijenjati kod ontologije te takve promjene, ukoliko zadovoljavaju određene uvjete, pohraniti. Promjene pohranjene u bilo kojem pregledniku unutar *Form panela* vidljive su i na ostalim preglednicima.



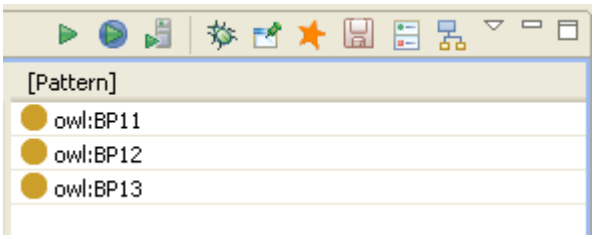
Slika 27. Form panel – Source code preglednik

Prozor primjeraka, članova klasa (*Instances*) nam prikazuje primjerke određene odabrane klase. Ti članovi sami po sebi su jedna vrsta klase koje nemaju podklasu, tj. imaju samo jednu, a to je već ranije spomenuta preddefinirana klasa *owl:Nothing* koja označuje propada li objekt nekoj običnoj klasi ili primjerku. Klase se inače označavaju žutim kružićem, a primjerci se označavaju ljubičastim ukošenim četverokutom. Prozor ima funkciju kreiranja i brisanja primjeraka, te mogućnost prikaza indirektnih primjeraka (npr. odabirom superklase vidimo primjerke njezinih podklasa).



Slika 28. Prozor s instancama klasa

TopBraid Composer sadrži još jednu relevantnu komponentu nutar svog sučelja, a to je prozor za upisivanje *SPARQL* upita. Nakon upisanog SPARQL koda, upit se pokreće zelenom strelicom uz mogućnost odabira pretraživanja samo ontološke baze ili ontološke baze s priključenim logički zaključenim vezama između klasa. Uz prostor za upis upita postoji i prostor za ispis rezultata. Rezultate je moguće pomoću dodatnih opcija eksportirati u nekom drugom formatu ili ih sačuvati za kasniju upotrebu.



Slika 29. Prozor za rezultate SPARQL upita i raznih opcija za konfiguraciju upita

4.3. Ontologija robotskog sustava

Ontologije služe za kreiranje i pohranu znanja o proizvoljnoj domeni. Unutar ontologije nalazi se formalna reprezentacija klasa sa pripadajućim svojstvima i međusobnim odosima. Vrlo je bitno kod ontologije da se izbjegne unošenje previše podataka o istoj stvari i smanji broj osoba koje u tu ontologiju unose podatke jer se time smanjuju preklapanja, razlike i druge greške. Prema tome, ontologija bi trebala biti što jednostavnija, tj. da ne sadrži prekompleksne logičke izraze. Drugi razlog za izradu jednostavne ontologije je i to što SPARQL upitni jezik radi s *RDF*-om i *RDFS*-om čije strukture ne podržavaju deskriptivnu logiku. Moguće je da korištenjem kompleksne logike zbog nedovoljno razvijenih software komponenti, koje su potrebne za smještaj komponenti ontologije, baza znanja pomoću SPARQL upita ne vrati očekivani rezultat. Kao alat za izradu ontologije izabran je *TopBraid Composer*, opisan u prijašnjem potpoglavlju. Cilj izrade ontologije je izrada baze znanja robotskog sustava pomoću koje će se SPARQL upitima moći izvlačiti potrebne informacije o samom radu sustava.

Na temelju odziva na *SPARQL* upit za pojedino radno mjesto, ontološka jezgra predlaže jedno ili više rješenja u vidu, tzv. uzoraka ponašanja (*BPs* – engl. *Behavioral Patterns*). Uzorak ponašanja je programski slijed instrukcija definiran algoritmom, te sadrži definiciju onoga što robot treba napraviti kao odziv na trenutno stanje okoline, uključujući provjeru odsutnosti/prisutnosti objekata od interesa, upravljanje stanjem signala zaustavnog mjesta u svrhu propuštanja/zadržavanja nosača proizvoda na liniji za sklapanje, i predefinirane i svrsishodne kretnje robota prilikom zamjene alata ili manipulacije objektima od interesa. [6]

Uzorci ponašanja su neki programi pohranjeni u robotu koji označavaju skup radnji koje robot izvršava. Obrazac ponašanja koji će robot izvršavati ovisi o trenutnom staju okoline, odnosno podacima koje senzori iz okoline skupljaju i šalju ontološkoj jezgri. Trenutak kada nosač dijelova stigne na zaustavno mjesto označava trenutak u kojem se provjere stanja senzora, te generira i šalje *SPARQL* upit. Kao reakciju na trenutačno stanje okoline, podataka koji senzori šalju, ontološka jezgra predlaže jedan ili više uzoraka ponašanja. Logička ovisnost stanja senzora (ulaza u sustav) i uzoraka ponašanja (izlaza iz sustava) je unaprijed određena i prikazana na tablici [Tablica 1.] vidljivo je da se jednom kombinacijom senzora može doći do rješenja s više obrazaca ponašanja, u tom se slučaju koriste sustavi za zaključivanje koji agentu daju točno određeni obrazac ponašanja.

	TC2	*TC1*	*CS*	*OB*	*VS2*	*VS1*	*SP*	*BP*
1	0	1	0	0		*0*	1	*BP1, BP2*
2	1	0	0	0		*0*	1	*BP1, BP2*
3	0	1	0	0		*Part, Assembly*	1	*BP1, BP2*
4	1	0	0	0		*Part, Assembly*	1	*BP1, BP2*
5	0	1	0	0		*Product*	1	*BP3*
6	1	0	0	0		*Product*	1	*BP3*
7	0	1	0	1		*0*	1	*BP4*
8	1	0	0	1		*0*	1	*BP4*
9	0	1	0	1		*Part, Assembly*	1	*BP5*
10	1	0	0	1		*Part, Assembly*	1	*BP5*
11	0	1	0	1		*Product*	1	*BP3*
12	1	0	0	1		*Product*	1	*BP3*
13	0	1	1	0		*0*	1	*BP1, BP2, BP3*
14	1	0	1	0		*0*	1	*BP1, BP2, BP3*
15	0	1	1	1		*0*	1	*BP2, BP4*
16	1	0	1	1		*0*	1	*BP2, BP4*
17	0	1	1	1		*Part, Assembly*	1	*BP2, BP5*
18	1	0	1	1		*Part, Assembly*	1	*BP2, BP5*
19	0	1	1	1		*Product*	1	*BP2, BP3*
20	1	0	1	1		*Product*	1	*BP2, BP3*

Tablica 1. Ovisnost stanja ulaza senzora i uzoraka ponašanja

Tablica je razvijena za jedno radno mjesto smješteno u definiranu okolinu, međutim, može se primjeniti i na više radnih mjesta (*Working Places*). Razvijeno je ukupno pet obrazaca ponašanja (*BP1...BP5*). Prema tablici vidljivo je da će za prvi slučaj kad senzori *TC2, CS,OB, VS1 i VS2* imaju vrijednost *false* a senzori TC1 i SP1 imaju vrijednost *true* ontološka jezgra modela agentu sugerirati obrazce ponašanja *BP1 i BP2*. Razvijeno je ukupno dvadeset kombinacija stanja ulaza senzora te ukupno devet različitih kombinacija izlaza uzoraka ponašanja.

Proces razvoja ontologije uključuje definiranje taksonomije. Kako bi se sačuvala kompatibilnost s ontologijama ostalih autora koje pokrivaju željeno područje, prvotno je potrebno istražiti već postojeće ontologije. Taksonomija ontološke jezgre treba objediniti sve sastavnice okoline koje su značajne za sustav. Svi elementi koji se nalaze u radnom prostoru smješteni su u klasu radnog prostora *owl:WorkingPlace*. Podklase radnog prostora su klase koje označuju rade prostore zasebnih modela sustava *(owl:WP1, owl:WP2)*.

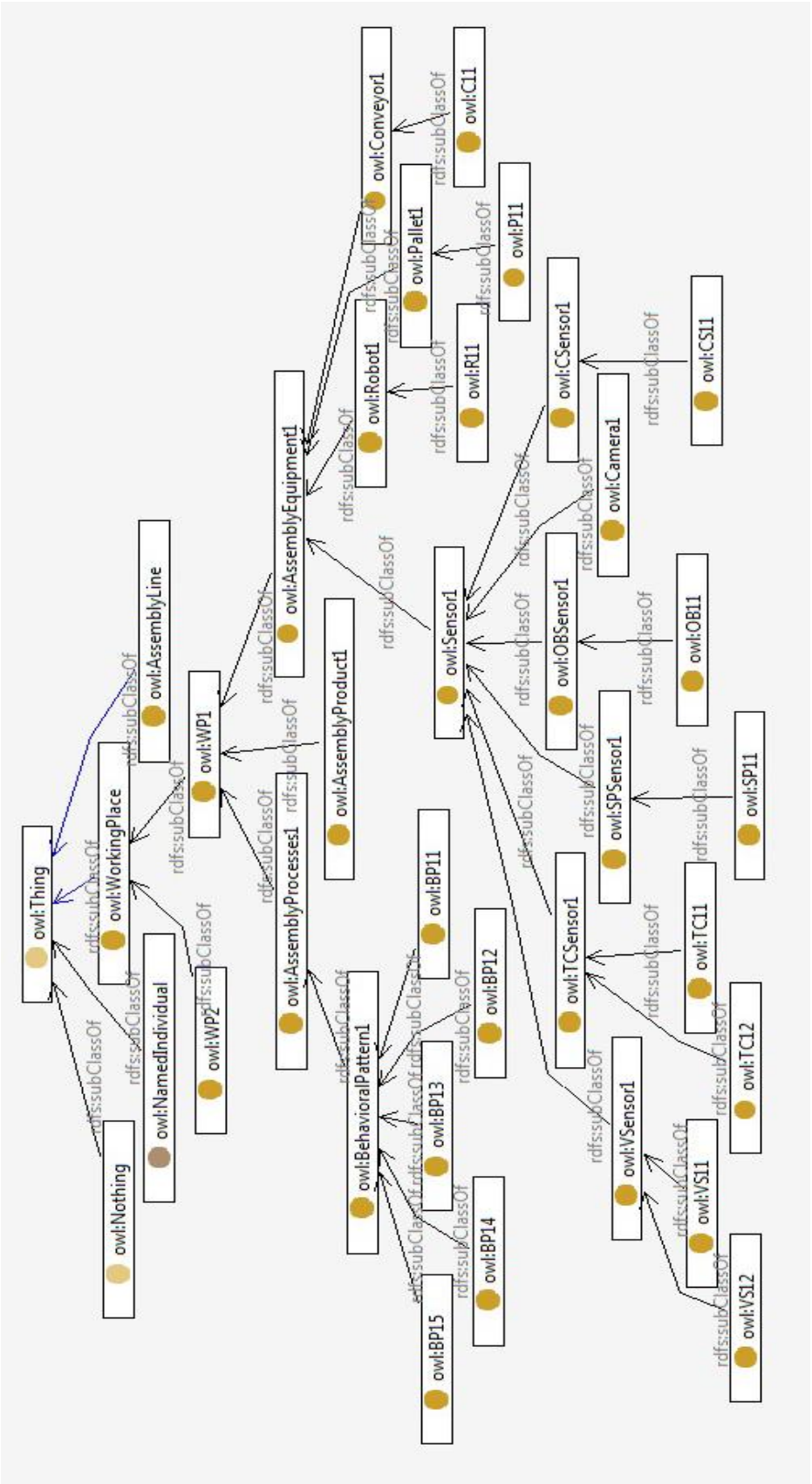
Osnovna podjela spoznajnog modela je: produkti sklapanja (*Assembly product → owl:AssemblyProduct1),* procesi sklapanja (*Assembly processes → owl:AssemblyProcesses1),* oprema za sklapanje (*Assembly Equipment → owl:AssemblyEquipment1).* Na kraju svake klase se nalazi broj koji označava da je ta klasa jedna od podklasi prvog radnog prostora (*owl:WP1*).

Unutar klase *owl:AssemblyProduct1* nalaze se podklase koje predstavljaju stanje proizvoda nad kojim se vrši montaža. Te podklase su: *owl:Part1*, *owl:Assembly1* i *owl:Product*. Klasa *owl:Part1* predstavlja jedan od elementa proizvoda za montažu, klasa *owl:Assembly* predstavlja barem dva spojena elementa koji ne sačinjavaju gotov proizvod, a klasa *owl:Product1* predstavlja gotovi proizvod.

Klasa *owl:AssemblyProcesses1* sadrži podklasu *owl:BehavioralPattern1* koja označuje i u kojoj su smješteni obrazci ponašanja koje je potrebno dobiti kao rješenje preko SPARQL upita po otprije određenim uvjetima. Klase koje predstavljaju pojedine obrzce ponašanja su *owl:BP11, owl:BP12, owl:BP13, owl:BP14 i owl:BP15.*

Senzori, robot i drugi elementi koji su sastavni dio opreme za montažu na radnom prostoru sadržani su u podklasama klase *owl:AssemblyEquipment1*. Podklase navede klase su *owl:Conveyor1, owl:Pallet1, owl:Robot1* i najzanimljivija klasa koja sadrži senzore *owl:Sensor1*. Klasa senzora sadrži instance senzora koji imaju postavljene vrijednsti svojstva *owl:hasValue* prema stvarnim stanjima senzora na radnom mjestu. Klase senzora su: *owl:Csensor1, owl:OBSensor1, owl:Spsensor1, owl:TCSensor1* i *owl:Vsensor1*. Stanja instanci navedenih klasa će se koristiti pri odabiru obrazca ponašanja.

Važan dio ontologije predstavljaju svojstva koja se dodjeljuju klasama. U navedenoj ontologiji koriste se razna svojstva pomoću kojih se na temelju zaključivanja stvaraju nove zaključne klase (*inferred classes)* i zaključni *RDF* trojke. Pomoću *RDF* trojki klase unutar ontologije se dodatno povezuju te se stvara ontologija koja se sastoji od osnovne ontološke jezgre i zaključenih *RDF* trojki.



Slika 30. Grafički prikaz taksonomije ontološke jezgre

5. PRETRAŽIVANJE RDF BAZE PODATAKA

RDF nam omogućava odlične načine spremanja modela i baze podataka, infrastrukture povezanih podataka nam omogućuju neograničene količine podataka za korištenje. Od kad postoje povezane baze podataka bazirane na *RDF*-u, postoje i programske biblioteke (engl. *progrmming libraries*) koje omogućuju postavljanje *RDF* trojki u strukturu podataka popularnih programskih jezika kako bi se moglo graditi aplikacije temeljene na tim postavljenim podacima. Kao što je vidljivo na relacijskim bazama podataka i u svijetu *XML* datoteka, programski jezik za izravno pretraživanje, kojem nije potrebno prethodno sastavljanje (*compiling*) kodova kako bi se izvršio, pojednostavljuje i olakšava korisnicima brže izrađivanje potrebnih aplikacija. Jedan takav jezik za pretraživanje *RDF* bazi podataka koji je ujedno korišten i u ovom radu, je *SPARQL*. Navedeni jezik za pretraživanje je i službena preporuka *W3C*-a (*World Wide Web Consortium*).

5.1. SPARQL 1.1.

SPARQL (SPARQL Protocol and RDF Query Language) je programski jezik za pretraživanje *RDF* trojki. Može se rei da je *SPARQL* za *RDF* baze podataka (semantički web), ono što je *SQL* za relacijske baze podataka. *SPARQL* može pretraživati i upravljati pohranjenim podacima u *RDF* formatu. *SPARQL* programski jezik je *RDF* (*Resource Description Framework*) grupa *DAWG* (*Data Access Working Group*) iz *W3C*-a proglasila standardom i smatra se jednom od ključnih tehnologija semantičkog weba. Od 15. siječnja 2008. godine, *SPARQL 1.0* postao je službena preporuka *World Wide Web* Konzorcija, a *SPARQL 1.1* od ožujka 2013. godine. Od početka implementacija *SPARQL* programskog jezika razna spremišta modela trojki (*Triplestores)* podržavaju *SPARQL* i samostalne alate kao što je *ARQ*. Počele su se pojavljivati i *SPARQL* dostupne točke (*SPARQL endpoints*) koje prihvaćaju *SPARQL* upite preko weba i vraćaju rezultate u željenom formatu. Do izlaza nove verzije *SPARQL*-a 1.1, baze podataka je bilo moguće samo pretraživati, bile su *read-only*, tek izlaskom nove verzije moguće je mijenjati baze podataka, dodavati ili brisati.

Za *SPARQL* 1.0 postoje tri specifikacijska dokumenta:

- *SPARQL Query Language for RDF* (*SPARQL* jezik za pretraživanje *RDF*-a) pokriva samu sintaksu pretrage. *QL (query language)* u nazivu označava ujedno područje kojim se korisnici najčešće i služe.

- *SPARQL Protocol for RDF* (*SPARQL* protokol za *RDF*) određuje na koji način će program slati *SPARQL* upite prema procesnim servisima i na koji način će server dati povratnu informaciju. Ova specifikacija se više odnosi na implementaciju *SPARQL* softvera nego programera koji zadaju upit za neku bazu podataka.
- *SPARQL Query Results XML Format* (*SPARQL* upiti rezultiraju *XML* formatima) opisuje jednostavni *XML* format koji koriste procesori upita pri vraćanju rezultata, odnosno povratne informacije su u *XML* formatu. Ukoliko se nekom procesoru zada upit i rezutat upita se dobije kao doument *XML* formata, taj dokument je lako moguće korištenjem *XSTL*-a ili nekog drugog *XML* alata pretvoriti u dokument bilo kojeg željenog formata.

Trud izrade *SPARQL*-a 1.1 urodio je dodavanjem nekoliko tehničkih dokumenata *SPARQL* kolekciji. Neki od njih će postati službene preporuke dok će se ostali smatrati pozadinskim informacijama (neće biti normativ). Neke od njih se odnose više na implementatore *SPARQL* softvera nego na programere koji upisuju upite nasprem bazama podataka.

- *The Federation Extensions* specifikacija nam opisuje način na koji jedan običan upit može dosegnuti informaciju iz više izvora. Time se olakšava izrada aplikacija kojima je velika prednost distribuiranost podataka.
- *The Update* specifikacija ključna je razlika između *SPARQL*-a 1.0 i *SPARQL*-a 1.1, čime se *SPARQL* od običnog programskog jezika za pretraživanje pretvorio u značajniji jezik kojim je moguće dodavati podatke, mijenjati ih i brisati iz baze podataka.
- *The Service Descriptin* opisuje način na koji korisnik programa može poslati upit samom sustavu *SPARQL*-a tražeći odgovor na pitanja o tome koje funkcije trenuta verzija podržava.
- *The Query Results JSON Format* opisuje *JSON* ekvivalent specifikacijskom dokumentu *Query Results XML Format*.
- *The Graph Store HTTP Protocol* proširuje *SPARQL*-ov protokol za komunikaciju između korisnika i *SPARQL* procesora u vezi grafova ili setova trojki. Omogućava više načina na koj se *HTTP*-u mogu odaslati informacija o tome koji graf da se priloži u bazu podataka. Također omogućava *HTTP*-u odašiljanje informacija o grafovima- koji se od njih iz baze podataka treba obrisati.

- *The Entailment Regimes* specifkacija opisuje kriterije za određivanje koju informaciju će *SPARQL* procesor uzeti u obzir pri utvrđivanju točnosti neke informacije. Ukoliko su podatak A i podatak B u takvom međusobnom odnosu da ukoliko je podatak A točan onda i podatak B mora isto tako biti točan, nečime je tu činjenicu potrebno i utvrditi. U navedenom slučaju je potrebno imati tehnologiju na principu *OWL*-ovog *SPARQL* procesora koji može odgonetnuti je li je podatak B stvarno točan. Postoji veliki broj informacija i time dolazi do zbrke kod odabira potrebnih informacija koje bi trebalo uzeti u obzir pri odlučivanju. Prema tome, navedena specifikacija nam govori koje informacije uzeti u obzir kod odlučivanja.
- *The SPARQL New Featres and Rationale* je dokument za nove dodatke i preporučje se za isusne korisnike *SPARQL*-a 1.0 koji iz *SPARQL*-a 1.1 žele izvući što više i što je brže moguće.

SPARQL je programski jezik za pretraživanje koji podosta funkcija dijeli sa sličnim programskm jezicima kao što su *XQUERY* i *SQL*. Osnovna ideja *SPARQL* jezika za pretraživanje je ta da se omogući pisanje običnog pitanja koje bi bilo nalik na podatke, gdje bi riječ s upitnikom unutar rečenice označavala pojam koji nas zanima. Kao i ostali jezici za pretraživanje relacijskih baza podataka, *SPARQL* nema namjeru oponašati sintaksu prirodnog jezika, ali koristi ideju da postavljeno pitanje može izgledati kao izjava, a riječ s upitnikom predstavlja pojam koji nas zanima.

5.2. Sintaksa SPARQL-a

Upit putem *SPARQL* jezika za pretraživanje funkcionira tako da se unutar upita upisuju *RDF* trojke gdje riječ s upitnikom predstavlja varijablu, pojam koji nas zanima. Pomoću varijable možemo spremati rezultat, koristiti varijablu u sljedećem redu pretrage ontolgije ili filtrirati rezultat. Varijablu nije potrebno unaprijed deklarirati, bitno je da se ispred željenog imena varijable (ime se odabere proizvoljno) nalazi upitnik (?).

SPARQL upit se sastoji od dva dijela i postavlja se kombinacijom više naredbi. U prvom dijelu *SPARQL* upita definiramo kakvu formu će imati naš upit. Naredbe u prvom dijelu upita mogu biti: *SELECT, CONSTRUCT, ASK* i *DESCRIBE*. Najčešće korištena je naredba *SELECT* kojom određujemo koje varijable želimo da nam se prikažu kao konačan rezultat. Pomoću drugih naredbi također se određuje prikaz rezultata upita.

Unutar upita, neke varijable nam služe za povezivanje trojki i nisu nam nužno bitne kod konačnog rezultata te je ovom naredbom moguće eliminirati nepotrebne varijable iz konačnih rezultata. Ukoliko ipak želimo da nam se prikažu sve varijable, onda umjesto pojedinačnog upisivanja varijabli možemo upisti zvjezdicu (*). Na slici [Slika 31.] može se vidjeti navedeni primjer gdje su *?subject* i *?object* tražene varijable, a upisanom zvjezdicom tražimo ispis svih varijabli koje se nalaze u upitu. Kao rješenje uipta dobit ćemo sve instance i klase koje odgovaraju zadanom upitu unutar naše ontologije.

```
SELECT *
WHERE {
  ?subject rdfs:subClassOf ?object .
}
```

Slika 31. Prikaz osnovnog SPARQL upita s naredbom SELECT

CONSTRUCT naredbom pomoću podataka dobivenih SPARQL upitom moguće je konstruirati važeću RDF trojku. Naredba *ASK* je najjednostavnija te se kao rješenje dobiva rezultat u obliku *true* ili *false* (točno ili netočno). Upit se postavlja kao tvrdnja te se upit vrši ispitivanjem točnosti te tvrdnje, tj. ispituje se postojanost i točnost RDF trojke. Naredba *DESCRIBE* koristi se za ispis podataka o izvoru za traženi pojam. Sve navedene naredbe osim naredbe *DESCRIBE,* kao drugi dio upita imaju nadovezanu naredbu *WHERE* kojom se oblikuje *SPARQL* upit. Kod naredbe *DESCRIBE* ona nije nužno potrebna. Naredba *WHERE* u sebi sadrži RDF trojke s varijablama pisane u *Turtle* obliku. Trojke se upisuju u vitičastu zagradu „{„ i „}“.Na kraju svake trojke potrbeno je postaviti točku (.). *URI*-e se postavlja između izlomljenih zagrada „<“ i „>“, *Qname* se piše bez ikakvih dodatnih znakova. Kod pisanja trojki za ponavljanje subjekta se postavlja točka-zarez (;) a za ponavljanje subjekta i predikata zarez (,). Kod pisanja SPARQL upita moguće je određivati i prefikse pomoću naredbe *PREFIX*. Navedena naredba piše se prije samog upita tako da se odabere prefiks, a izvor postavi u izlomljene zagrade. Ukoliko pomoću upita dobimo više identičnih rezultata, takve rezultate možemo filtrirati pomoću dodatka *DISTINCT* koji se pridodaje naredbi *SELECT*. Dodatkom *DISTINCT* (različito) se ostavlja po jedan rezultat dok se svi identični (ponavljajući) filtriraju, tako da je nemoguće dobiti dva ista rješenja, to joj je jedina mogućnost i nema drugu primjenu. Naredba *DISTINCT* koja služi za filtriranje nema veze s naredbom *FILTER* koja također služi za filtriranje rezulta.

Nakon što je postavljen upit, u kod upita možemo ubaciti i naredbe za upravljanje rješenjima. Možemo određivati redoslijed ispisa rezultata naredbom *GROUP BY* ili *ORDER BY*. Rezultate potom možemo ograničavati naredbama *LIMIT* ili filtrirati već spomenutom naredbom *FILTER*. Navedene naredbe postavljaju se u *SPARQL* upit nakon osnovnog upita, tj. nakon izvršenja naredbe *WHERE*. Unutar jednog upita možemo postaviti dodatni upit čija će rješenja biti sastavni dio glavnog upita. Za to nije potrebna nikakva dodatna funkcija, već se unutar naredbe WHERE u vitičastim zagradama upisuju nove naredbe *SELECT* i *WHERE* ili *CONSTRUCT* i *WHERE*, ovisi koji su nam podaci potrebni. Ovakav način pisanja upita naziva se *Subquery* (podupit). Preostali primjeri naredbi vidljivi su u tablici [Tablica 2.].

Kategorija	Funkcije / Operatori	Primjeri
Logical & Comparisons	!, &&, ||, =, !=, <, <=, >, >=, IN, NOT IN	?hasPermit || ?age < 25
Conditionals *(SPARQL 1.1)*	EXISTS, NOT EXISTS, IF, COALESCE	NOT EXISTS { ?p foaf:mbox ?email }
Math	+, -, *, /, abs, round, ceil, floor, RAND	?decimal * 10 > ?minPercent
Strings *(SPARQL 1.1)*	STRLEN, SUBSTR, UCASE, LCASE, STRSTARTS, CONCAT, STRENDS, CONTAINS, STRBEFORE, STRAFTER	STRLEN(?description) < 255
Date/time *(SPARQL 1.1)*	now, year, month, day, hours, minutes, seconds, timezone, tz	month(now()) < 4
SPARQL tests	isURI, isBlank, isLiteral, isNumeric, bound	isURI(?person) || !bound(?person)
Constructors *(SPARQL 1.1)*	URI, BNODE, STRDT, STRLANG, UUID, STRUUID	STRLANG(?text, "en") = "hello"@en
Accessors	str, lang, datatype	lang(?title) = "en"
Hashing *(1.1)*	MD5, SHA1, SHA256, SHA512	BIND(SHA256(?email) AS ?hash)
Miscellaneous	sameTerm, langMatches, regex, REPLACE	regex(?ssn, "\\d{3}-\\d{2}-\\d{4}")

Tablica 2. SPARQL naredbe

5.3. SPARQL upiti

Kompliciranа ontologija u raznim slučajevima može biti nekompatibilna s postojećim hardverom te je iz tog razloga potrebno razviti jednostavnu ontološku jezgru. Budući da je taj način do sada uvelike isproban u praksi, logički izrazi bi većinom trebali biti implementirani untar *SPARQL* upita. Kod prvog primjera *SPARQL* upita originalna ontologija je u nekim svojim dijelovima preuređena kako bi bilo lakše konstruirati upit.

Kao što je već navedeno, ontologija se sastoji od nekoliko podjela klasa od kojih su nama zanimljive: *owl:Sensor1* koja sadrži razne podklase koje predstavljaju pojedine senzore i *owl:BehavioralPattern1* koja sadrži podklase koje predstavljaju obrazac ponašanja (redoslijed izvedbi operacija u montažnom sustavu). Različiti podaci dobiveni sa strane senzora pokreću različite obrazce ponašanja, zbog toga je potrebno razviti sustav unutar ontologije koji će za točno određene izlazne veličine senzora na upit dati željeni obrazac ponašanja ili više njih. Razvijeni sustav je jednostavan i razvijen u kratkom roku koristeći jednostavnu ontologiju klasa i podklasa, a rezultat se dobiva pomoću *SPARQL query* upita.

Dakle, radi se o jednostavnoj *input-output* logici. Svi izlazi senzora su binarni (1/0 , *true/false*) osim vizijskog senzora koji ima mogućnost detektiranja više stanja. Izlaz vizijskog senzora može biti: *0/false, product, part* ili *assembly*, vrsta izlaznih podataka iz vizijskog senzora je „*string*“ što predstavlja neki znakovni niz koji može a i ne mora imati posebno značenje. Kod ostalih senzora izlaz je *true* ili *false*, odnosno 1 ili 0 te se u ontologiji za vrijednost senzora stavlja „*boolean*“ vrsta podatka.

U tablici [Tablica 1.] vidljiv je prikaz tablice odnosa ulaza i izlaza na temelju koje je potrebno napraviti SPARQL upit. Vidljivo je da tablica sadrži dvadeset kombinacija od kojih imamo deset različitih rješenja. Ukoliko se promjene samo vrijednosti senzora *TC1* i *TC2*, neće doći do promjene vrijednosti na izlazu te se u sljedećim *SPARQL* upitima vrijednosti senzora *TC1* i *TC2* ne uzimaju u obzir. Kod prva dva primjera pretraživanja ontologije pri pretraživanju ne uzimaju se u obzir ni promjene senzora *SP1* i *VS2*, budući da su njihove vrijednosti konstantne kod svih kombinacija izlaza (*VS2* uvijek ima vrijednost *false*, dok *SP* uvijek ima vrijednost *true*). Iz tih razloga broj kombinacija izlaza pri izradi *SPARQL* upita je smanjen na deset, pri čemu se u obzir uzimaju najvažnije vrijednosti senzora *OB, CS i VS1*. Treći primjer SPARQL upita daje i rješenje u obliku *string*-a ukoliko postavljene vrijednosti senzora nisu navedene unutar predočene tablice ili dođe do greške pri unosu podataka.

5.3.1. Prvi primjer SPARQL upita

U opisu ontologije objašnjena su stanja senzora, međutim, za ovaj tip SPARQL upita unutar ontologije postavljena su neka nova svojstva. Postavljena svojstva prezentiraju vrijednosti senzora i imaju stanja *true*, *false*, *part*, *assembly* ili *product*. Kao što je ranije naglašeno, skoro svako svojstvo koje je dio OWL ontologije ima domenu (engl. *Domain*) i doseg (*engl. Range*).

Za svako novo postavljeno svojstvo *owl:imaVrijednostOB11*, *owl:imaVrijednostCS11*, *owl:imaVrijednostSP11*, *owl:imaVrijednostTC11*, *owl:imaVrijednostTC12* definiran je tip podataka *boolean*. Dotična svojstva su specifična za svaki senzor posebno (npr. svojstvo *owl:imaVrijednostCS11* ima kao domenu klasu senzora *owl:CS11)*. Budući da vizijski senzor vraća vrijednost i tipa *string*, svojstvima *owl:imaVrijednostVS11* i *owl:imaVrijednostVS12* je dodjeljen pripadajući tip podataka. Sva svojstava senzora su u ontologiji postavljena kao podsvojstva svojstvu *owl:imaVrijednosti*.

```
owl:vs11
      a       owl:VS11 ;
      owl:imaVrijednostVS11
              "product"^^xsd:string .
```

Slika 32. Definicaija za vizijski senzor tipa string

```
owl:cs11
      a       owl:CS11 ;
      owl:imaVrijednostCS11
              "true"^^xsd:boolean .
```

Slika 33. Definicija za kapacitivni senzor tipa boolean

Iz tablice je vidljivo da se obrazac ponašanja 1 (*BP1„Behavioral Pattern 1“,* u ontologiji owl:*BP11)* aktivira/pokreće u šest slučajeva (1, 2, 3, 4, 13 i 14 red). Za svaki slučaj (kombinaciju izlaza senzora) napravljena je izvedena („inferred“) klasa kojoj je BP1 podklasa. „Inferred“ klasa (nazovimo je *Inferred1*) je definirana kao klasa (*a owl:Class*) te svojstvom *owl:isActivatedBy.* Svojstvo *owl:isActivatedBy* nema određene granice („*domain*“ i „*range*“) jer su mu granice izvedene klase (od kojih bi „*domain*“ bio tip klase *Inferred1*, a „*range*“ neka nova izvedena klasa).

Dakle, subjekt predikata *owl:isActivatedBy* je tip klase *Inferred1* dok je objekt neka nova izvedena klasa (nazovimo ju *Inferred2*) definirana ograničenjem (*restriction*). Ograničenje u ovom slučaju nije tipično kao kod ostalih ontologija, modificirano je, nema aktivno značenje te je namijenjeno potrebi ovog slučaja. Koristi se isključivo za povezivanje obrasca ponašanja s vrijednostima senzora. Izvedena klasa *Inferred1* se dakle sastoji od niza klasa *Inferred2* povezanih međusobno svojstvom *owl:isActivatedBy*. Dakle, svaka klasa *Inferred2* u sebi sadrži ograničenje s jednim od senzora te njegovom vrijednošću koja odgovara tablici ulaza i izlaza (prema tablici senzor *cs11* za obrazac ponašanja BP1 ima vrijednost „*false*“). Nakon što se za obrazac ponašanja unese prva kombinacija vrijednosti (sve klase ograničenja za odgovarajuće senzore) može se unijeti druga kombinacija (npr. drugi red). Izvedene klase s podklasama ograničenja će nam kasnije služiti za pretraživanje obrazaca ponašanja pomoću *SPARQL upita*.

```
owl:BP11
      a       owl:Class ;
      rdfs:subClassOf owl:BehavioralPattern1 ;
      rdfs:subClassOf
              [ a       owl:Class ;
                owl:isActivatedBy
                        [ a       owl:Restriction ;
                          owl:hasValue "true"^^xsd:boolean , "false"^^xsd:boolean ;
                          owl:onProperty owl:imaVrijednostTC11
                        ] ;
                owl:isActivatedBy
                        [ a       owl:Restriction ;
                          owl:hasValue "true"^^xsd:boolean ;
                          owl:onProperty owl:imaVrijednostSP11
                        ] ;
                owl:isActivatedBy
                        [ a       owl:Restriction ;
                          owl:hasValue "false"^^xsd:boolean ;
                          owl:onProperty owl:imaVrijednostCS11
                        ] ;
                owl:isActivatedBy
                        [ a       owl:Restriction ;
                          owl:hasValue "assembly"^^xsd:string , "part"^^xsd:string , "false"^^xsd:string ;
                          owl:onProperty owl:imaVrijednostVS11
                        ] ;
                owl:isActivatedBy
                        [ a       owl:Restriction ;
                          owl:hasValue "false"^^xsd:string ;
                          owl:onProperty owl:imaVrijednostVS12
                        ] ;
                owl:isActivatedBy
                        [ a       owl:Restriction ;
                          owl:hasValue "true"^^xsd:boolean , "false"^^xsd:boolean ;
                          owl:onProperty owl:imaVrijednostTC12
                        ] ;
                owl:isActivatedBy
                        [ a       owl:Restriction ;
                          owl:hasValue "false"^^xsd:boolean ;
                          owl:onProperty owl:imaVrijednostOB11
                        ]
              ] ;
```

Slika 34. Prikaz klase uzorka ponašanja BP1 s jednom od kombinacija izlaza senzora

Unos podataka, izvedenih klasa, je zbog jednostvnosti izveden preko prozora s izvornim kodom u obliku „*turtle*“. Iako je vidljivo da je originalni kod podosta velik, direktan unos koda (u *turtle* obliku) je moguće skratiti postavljanjem određenih zagrada i zareza, nakon aktiviranja koda program ga prilagodi svojim potrebama i proširi. Postoji podosta ponavljanja kodova te je stoga pri unosu podataka i vrijednosti senzora potrebno oprezno pristupiti da ne bi došlo do zamjene vrijednosti. Prikazani SPARQL kod je prvi pokušaj pretraživanja baze podataka te kao prvi pokušaj prilično odsupa od konačnog rješenja, odnosno od najpraktičnijeg pretraživanja baze podataka.

```
SELECT  ?Pattern ?tc1 ?tc2 ?cs1 ?ob1 ?vs1 ?vs2 ?sp1
WHERE {
owl:tc11 owl:imaVrijednostTC11 ?tc1 .
owl:tc12 owl:imaVrijednostTC12 ?tc2 .
owl:cs11 owl:imaVrijednostCS11 ?cs1 .
owl:ob11 owl:imaVrijednostOB11 ?ob1 .
owl:vs11 owl:imaVrijednostVS11 ?vs1 .
owl:vs12 owl:imaVrijednostVS12 ?vs2 .
owl:sp11 owl:imaVrijednostSP11 ?sp1 .

?PK owl:isActivatedBy
     [ a      owl:Restriction ;
       owl:hasValue ?tc1 ;
       owl:onProperty owl:imaVrijednostTC11
     ] .
?PK owl:isActivatedBy
     [ a      owl:Restriction ;
       owl:hasValue ?ob1 ;
       owl:onProperty owl:imaVrijednostOB11
     ] .
?PK owl:isActivatedBy
     [ a      owl:Restriction ;
       owl:hasValue ?vs2 ;
       owl:onProperty owl:imaVrijednostVS12
     ] .
?PK owl:isActivatedBy
     [ a      owl:Restriction ;
       owl:hasValue ?vs1 ;
       owl:onProperty owl:imaVrijednostVS11
     ] .
?PK owl:isActivatedBy
     [ a      owl:Restriction ;
       owl:hasValue ?sp1 ;
       owl:onProperty owl:imaVrijednostSP11
     ] .
?PK owl:isActivatedBy
     [ a      owl:Restriction ;
       owl:hasValue ?tc2 ;
       owl:onProperty owl:imaVrijednostTC12
     ] .
?PK owl:isActivatedBy
     [ a      owl:Restriction ;
       owl:hasValue ?cs1 ;
       owl:onProperty owl:imaVrijednostCS11
     ] .

?Pattern rdfs:subClassOf ?PK .
?Pattern rdfs:subClassOf owl:BehavioralPattern1
 }
```

Slika 35. Prikaz prvog primjera SPARQL upita

Nakon što su za sve obrasce ponašanja unesene sve vrijednosti izlaza senzora, pristupa se izradi *SPARQL* upita. *SPARQL* (engl. *SPARQL Protocol And RDF Query Language*) je jezik za dohvat RDF podataka iz željene ontologije, a radi na principu podudaranja uzoraka (*triplova* unutar ontologije). U *SPARQL* su prvo uneseni željeni izlazi upita: obrasci ponašanja, vrijednosti senzora. U upit je moguće po želji unijeti i druge vrijednosti iz ontologije. *SPARQL* query (*SPARQL* upit) se zasniva na izvedenim klasama s ograničenjima koje on po principu podudaranja pretražuje. Budući da su izlazne vrijednosti senzora unesene u ontologiju kao varijable, tako ih je potrebno unijeti i u *SPARQL query* pomoću trojki.

Unos varijabli izlaza senzora u *SPARQL query* nam omogućuje da jednom izrađen upit više ne moramo mijenjati iako se vrijednosti izlaza senzora konstantno mijenjaju (upit je potrebno mijenjati jedino ukoliko dođe do dodavanja još jednog senzora ontologiji, odnosno ukoliko se on želi pridodati dotičnoj tablici i upitu o odnosu ulaza i izlaza).

Nakon unosa varijabli stanja izlaza senzora unose se trojke koje su slične izvedenoj klasi s ograničenjem samo što umjesto vrijednosti (*true ili false*) ima varijablu izlaza senzora koja je unaprijed definirana u ontologiji. Svaka trojka s ograničenjem u *SPARQL*-u se postavlja kao objekt predikata *owl:isActivatedBy*, a subjekt je varijabla *?PK* koja predstavlja jedanu od izvedenih klasa tipa *Inferred1* (predstavlja i jednu od kombinacija izlaza). Nakon unosa svih *trojki* s ograničenjima (za dotični primjer 7, za jednu vrijednost svakog senzora) postavlja se upit za obrazac ponašanja koji je podklasa dobivenoj kombinaciji odnosno izvedenoj klasi. Za svaki slučaj može se unijeti i konstatacija za upit da je traženi obrazac ponašanja podklasa *BehavioralPattern1* (superklasa svim obrazcima ponašanja).

Postavljanjem senzora TC1 i SP na vrijednost *true* te svih ostalih na vrijednost *false*, *SPARQL* upitom na [Slika 35.] dobiju se željena rješenja. Obrazci ponašanja koji odgovaraju traženom stanju senzora su BP1 i BP2 (prva od kombinacija senzora prema tablici). Postavljanjem drugih vrijednosti senzora također je dobiven očekivani rezultat.

[Pattern]	tc1	tc2	cs1	ob1	vs1	vs2	sp1
owl:BP11	true	false	false	false	false	false	true
owl:BP12	true	false	false	false	false	false	true

Tablica 3. Rezultat SPARQL upita

5.3.2. Drugi primjer SPARQL upita

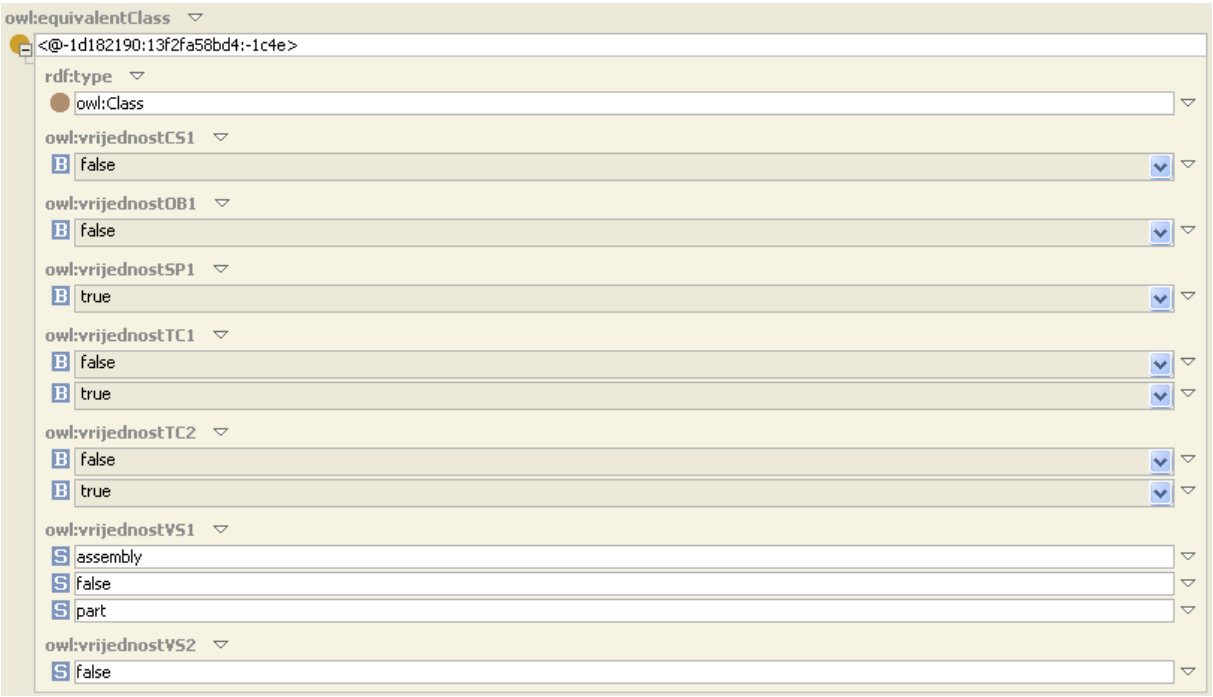
U prvoj verziji SPARQL upita postavljena su nova svojstva za svaki senzor posebno te se njih povezivalo ograničenjima (*restriction*) vrijednosti. Takav način pretraživanja zahtijevao je podosta preinaka ontologije u odnosu na početnu (dodavanje novih svojstava, raspisivanje trojki ograničenja za svaki slučaj). Veliki dio logike nije se vršio unutar samog *SPARQL* koda već je bio definiran unutar ontologije i zasnivao se na pretraživanju uvjeta ogrančenja. Cilj je što manje mijenjati početni ontološki model kako bi sama ontološka jezgra ostala što jednostavnija.

U drugom primjeru SPARQL upita izbacila su se ograničenja, a nova svojstva pridodala obrazcima ponašanja. Drugi način pretraživanja se također bazira na uvedenim svojstvima koja predstavljaju logiku unutar ontologije pomoću koje se ontologija pretražuje. Dodana su svojstva: *owl:vrijednostCS1*, *owl:vrijednostOB1*, *owl:vrijednostSP1*, *owl:vrijednostTC1*, *owl:vrijednostTC2*, *owl:vrijednostVS1*, *owl:vrijednostVS2* i smještena su pod svojstvo *owl:vrijednost*. Svako svojstvo (osim *owl:vrijednost*) odnosi se na pojednini senzor, svojstvo *owl:vrijednostSP1* za senzor *SP1* itd. Svojstva su skoro identična kao kod prijašnjeg primjera. Razlika je u tome što svojstva nisu dodijeljena senzorima već obrazcima ponašanja.

```
owl:BP11
      a       owl:Class ;
      rdfs:subClassOf owl:BehavioralPattern1 ;
      owl:equivalentClass
              [ a       owl:Class ;
                owl:vrijednostCS1 "true"^^xsd:boolean ;
                owl:vrijednostOB1 "false"^^xsd:boolean ;
                owl:vrijednostSP1 "true"^^xsd:boolean ;
                owl:vrijednostTC1 "true"^^xsd:boolean , "false"^^xsd:boolean ;
                owl:vrijednostTC2 "true"^^xsd:boolean , "false"^^xsd:boolean ;
                owl:vrijednostVS1 "false"^^xsd:string ;
                owl:vrijednostVS2 "false"^^xsd:string
              ] ;
      owl:equivalentClass
              [ a       owl:Class ;
                owl:vrijednostCS1 "false"^^xsd:boolean ;
                owl:vrijednostOB1 "false"^^xsd:boolean ;
                owl:vrijednostSP1 "true"^^xsd:boolean ;
                owl:vrijednostTC1 "true"^^xsd:boolean , "false"^^xsd:boolean ;
                owl:vrijednostTC2 "true"^^xsd:boolean , "false"^^xsd:boolean ;
                owl:vrijednostVS1 "assembly"^^xsd:string , "part"^^xsd:string , "false"^^xsd:string ;
                owl:vrijednostVS2 "false"^^xsd:string
              ] .
```

Slika 36. Prikaz definicije obrazca ponašanja BP1

Navedenim promjenama u odnosu na prvi primjer manje se mijenja osnovna ontologija (promjena je samo kod obrasca ponašanja). Stanja senzora su definirana pomoću svojstva *owl:hasValue* i imaju već otprije navedene vrijednoati *true*, *false*, *part*, *assembly* ili *product.* Svaka klasa koja predstavlja obrazac ponašanja definirana je kao ekvivalentna klasi koja sadrži svojstva s odgovarajućim stanjima senzora. Ekvivalentne klase nemaju naziv i definirane su kao klase s novim svojstvima. Te klase u stvari predstavljaju jednu kombinaciju stanja senzora kojoj odgovara obrazac ponašanja.



Slika 37. Prikaz jedne od uvedenih ekvivalentnih klasa (u form prikazu)

SPARQL upit se zasniva na navedenim ekvivalentnim klasama. Kao i u prvom primjeru, trenutne vrijednosti senzora unesene su u *SPARQL* upit te se pomoću njih i ekvivalentnih klasa vrši pretraga. Pretržuje se koji obrazci ponašanja imaju ekvivalentnu klasu s trenutnim stanjima senzora. Na ovaj način promjene ontologije smanjene su na minimum a smanjen je i sam kod *SPARQL* upita. Na slici [Slika 38.] vidljivo je da se *SPARQL* upit sastoji od dva dijela. Prvi dio je unos trenutnih stanja senzora, a drugi dio predstavlja ekvivalentnu klasu koja se pretražuje. Vidljivo je da je novi SPARQL upit mnogo jednostavniji od prethodnog uz manje preinake osnovne ontologije sustava.

```
SELECT ?BP ?tc1 ?cs1 ?ob1 ?vs1
WHERE {

owl:tc11 owl:hasValue ?tc1 .
owl:tc12 owl:hasValue ?tc2 .
owl:cs11 owl:hasValue ?cs1 .
owl:ob11 owl:hasValue  ?ob1 .
owl:vs11 owl:hasValue  ?vs1 .
owl:vs12 owl:hasValue  ?vs2 .
owl:sp11 owl:hasValue  ?sp1 .

?BP owl:equivalentClass  [ a      owl:Class ;
            owl:vrijednostCS1 ?cs1;
            owl:vrijednostOB1 ?ob1 ;
            owl:vrijednostSP1 ?sp1 ;
            owl:vrijednostTC1 ?tc1 ;
            owl:vrijednostTC2 ?tc2 ;
            owl:vrijednostVS1 ?vs1 ;
            owl:vrijednostVS2 ?vs2
        ]
}
```

Slika 38. Prikaz SPARQL upita (drugi primjer)

Postavljanjem stanja senzora tako da odgovaraju zadnjem (dvadesetom) redu tablice te pokretanjem SPARQL upita dobivamo željeni rezultat. Daljnjim isprobavanjem kombinacija stanja senzora također se kao rješenja upita dobiju predviđeni obrasci ponašanja.

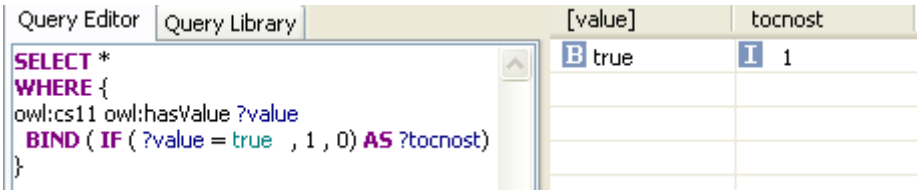
[BP]	tc1	cs1	ob1	vs1
owl:BP12	true	true	true	product
owl:BP13	true	true	true	product

Tablica 4. Rezultat SPARQL upita (drugi primjer)

5.3.3. Treći primjer SPARQL upita

Kod treće verzije SPARQL upita sva logika smještena je unutar samog SPARQL koda te su se eliminirale promjene koje su postojale kod prethodna dva primjera (uvođenje novih svojstava za pomoć pri pretražvanju). Upit se bazira na *BIND AS* i *IF* nardbama.

Na slici [Slika 39.] je vidljiv jednostavni primjer kombinirane uptrebe nardbi *BIND AS* i *IF*. Ukoliko je stanje senzora *CS11 true* tada će se varijabli *?tocnost* dodijeliti jedinica (1), ukliko je stanje senzora neka druga vrijednost varijabli će biti dodijeljena nula (0). U rezultatima vidimo da je trenutna vrijednost (*value*) *true* i sukladno tome točnost je potvrđena. Navedene naredbe osnova su trećeg primjera *SPARQL* upita



Slika 39. Prikaz osnovnog korištenja naredbe BIND AS i IF

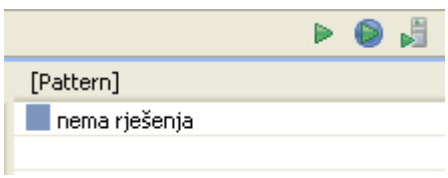
Unos vrijednosti stanja senzora identičan je kao i kod prethodna dva primjera. Umjesto pretraživanja ontološke jezgre na način da se pretražuju svojstva ili ograničenja klasa, u ovom primjeru se direktno pretražuju vrijednosti pojedinog senzora. Sva logika, koji će obrazac ponašanja biti dobiven kao rezultat pretrage, smještana je unutar *SPARQL* upita. *SPARQL* upitom pretražuju se vrijednosti senzora jedan za drugim te uspoređuju na način zapisan u kodu. Treći primjer SPARQL upita zbog implementirane logike bitno je drugačiji od prva dva primjera i predstavlja konačno rješenje u vezi postavljanja upita ontološkoj jezgri.

```
SELECT    DISTINCT ?Pattern
WHERE {
owl:tc11 owl:hasValue ?tc1 .
owl:tc12 owl:hasValue ?tc2 .
owl:cs11 owl:hasValue ?cs1 .
owl:ob11 owl:hasValue  ?ob1 .
owl:vs11 owl:hasValue  ?vs1 .
owl:vs12 owl:hasValue  ?vs2 .
owl:sp11 owl:hasValue  ?sp1 .

BIND(IF(?vs1="assembly", owl:Komb8, IF(?vs1="false", owl:Komb7, owl:Komb6)) AS ?X6)

BIND(IF(?vs1="product", owl:Komb9,IF(?vs1="part", owl:Komb8, ?X6))AS ?X3)

BIND(IF(?vs1="false", owl:Komb3 , IF(?vs1="part",owl:Komb4,
 IF(?vs1="assembly", owl:Komb4,  IF( ?vs1="product", owl:Komb5 , owl:Komb6)))) AS ?X4)

BIND(IF(?vs1="false", owl:Komb6 , owl:Komb6) AS ?X1)

BIND(IF(?vs1="false", owl:Komb1 , IF(?vs1="part", owl:Komb1,
IF(?vs1= "assembly", owl:Komb1, IF(?vs1="product", owl:Komb2, owl:Komb6)))) AS ?X5)

BIND(IF(?cs1=true , ?X1 , ?X5) AS ?X2)

BIND (IF(?ob1=true,IF(?cs1=true, ?X3, ?X4) , ?X2) AS ?Komb )

?BP rdfs:subClassOf ?Komb

BIND (IF(?cs1=true && ?ob1=false && ?vs1 != "false" && ?Komb = owl:Komb6, false,true ) AS ?D1)
BIND(IF (?tc1!=?tc2 && ?sp1 = true && ?vs2 = "false", true, false) AS ?D2)
BIND (IF(?vs1="false" || ?vs1 ="product" || ?vs1 ="part" || ?vs1 = "assebly" , true, false) AS ?D3)
BIND( IF(?D1 = true && ?D2 =true && ?D3=true, ?BP,"nema rješenja" ) AS ?Pattern)
}
```

Slika 40. Prikaz SPARQL upita (treći primjer)

Na slici [Slika 40.] prikazan je treći primjer *SPARQL* upita. U kodu se koriste pridodane klase kombinacija koje predstavljaju jednu od kombinacija obrazaca ponašanja. Tako klasa *owl:Komb1* ima podklase *owl:BP1* i *owl:BP2* koje predstavljaju obrazce ponašanja *BP1* i *BP2* koji se prema zadanoj tablici nalaze u prvom redu. Vidljivo je da se kod prve kombinacije naredbi *BIND AS i IF* provjerava je li vrijednost senzora *VS1 assembly* (vraijabla *?vs1*). Ukoliko je tvrdnja točna, varijabli *?X6* će se dodijeliti kombinacija obrazaca ponašanja osam (*owl:Komb8*). Ukoliko je tvrdnja netočna, varijabli je dodijeljeno rješenje iz nove *IF* naredbe koja je postavljena unutar postojeće. Kasnije se novim naredbama kombinacija obrazaca ponašanja dodjeljuje novoj varijabli *?X3*. U trenutku kad je varijabli ?X3 dodijeljena neka vrijednost, ostalim varijablama su također dodijeljene vrijednosti koje, ukoliko se ne zadovolje ostali uvjeti neće imati ulogu u stvaranju konačnog rješenja SPARQL upita. Kod provjere vrijednosti *OB1* (varijable *?ob1*) senzora u osnovnu naredbu *IF* postoji i podnaredba *IF* koja provjerava stanje senzora *CS1* (varijable *?cs1*). Podnaredbom se postavlja potvrdna varijabla za *IF* funkciju varijable *?ob1*. Ukoliko varijabla *?cs1* sadži vrijednost true, kao potvrdna vrijednost bit će postavljena varijabla *?X3*, u suprotnom to će biti varijabla *?X4*. Time je definirana potvrdna vrijednost naredbe *IF* u slučaju varijable *?ob1*, u slučaju netočnosti tvrdnje postavljena je varijabla *?X2* koja je definirana na sličan način kao i varijabla *?X3*. Pošto su obje vrijednosti senzora (*CS1* i *OB1*) *true*, konačana varijabla *?Komb* dobiva vrijednost *owl:Komb8*. Pomoću trojki unutar ontologije klasa *owl:komb8* se pretražuje za podklasama koje predstavlaju gotovo rješenje.

Obješnjeni dio koda predstavlja logiku za pronalažnje obrazaca ponašanja uzimajući u obzir vrijednosti senzora *OB1*, *CS1* i *VS1*. Ostali senzori ne utječu na izbor obrazaca ponašanja, ali ukoliko se njihove vrijednosti razlikuju od one tablici, cijeli upit neće biti točno koncipiran, odnosno rezultat će biti netočan. Zato je pridodana logika koja određuje postoj li uopće rješenje za postavljne vrijednosti senzora. U kodu je postavljeno da ukoliko su vrijednosti senzora *TC1* i *TC2* identične, vrijednost senzora *SP1* različita od *true*, a vrijedenost senzora *VS2* različita od *false*, neće biti moguće dobiti rješenje, odnosno *SPARQL* upit će kao rezultat izbacivati „*nema rješenja*“.



Slika 41. Rezultat upita gdje ne postoji traženo rješenje

6. KORISNIČKO SUČELJE

Kako bi ontologija bila vidljiva svim korisnicima, potrebno je izgraditi određeno sučelje. Najjedstavniji način je postaviti sučelje tako da je ontologiju moguće proučavati pomoću običnih internet preglednika. Sučelje ne treba biti komplicirano nego jednostavno, s ukomponiranim svim potrebnim dodacima za pregled i pretraživanje ontologija. Prema tome, cilj izrade sučelja za pregled ontologije pomoću interenet preglednika služi kako bi se uneseno znanje moglo pretraživati i proučavati pomoću interneta preko bilo kojeg računala. Struktura sučelja mora biti jednostavna i dobro dizajnirana kako bi se olakšalo njegovo korištenje. Sučelje je izrađeno pomoću *TopBraid* alata *Ensemble.* Kod izrade sučelja za pregled i pretraživanje ontologije potrebno je odrediti sljedeće:

- način pregleda ontologije (potrebno je izabrati odgovarajući način za prikaz stabla klasa i prikaz podataka klasa i instanci)
- način pretraživanja ontologije (potrebno je u sučelje ugraditi opciju za unos SPARQL upita i prozore za obične upite pomoću pojmova)
- izgled sučelja (smještaj preglednika, pretraživača, hijerarhije klasa u sučelju kako bi cijelokupno sučelje bilo estetski ljepše)

6.1. TopBraid Ensamble

Između nekoliko alata koji služe za izradu sučelja za pregled i pretraživanje ontologija, *TopBraid Ensemble* je izabran kao najprikladniji za izradu sučelja za internet preglednik (u korist *TopBraid Ensemblea* ide i to da je i ontologija sustava rađena u *TopBraid* paketu te je zbog toga prikladan za njezin prikaz). *TopBraid Ensemble* je jedan od tri alata koji se nalaze u *TopBraid* paketu (*TopBraid Suite)* tvrtke *TopQuadrant* koji služe za izradu, oblikovanje i prikaz ontologija.

TopBraid Ensemble je fleksibilni alat za izradu i prikaz aplikacija izrađenih pomoću *TopBraid Composera* (već spomenutog alata za izradu ontologija). Korisničko sučelje pokreće se pomoću internet preglednika koji mora u sebi sadržavati instalirani *Adobe Flash (Adobe Flash* je programska platforma koja služi za prikaz vektorske grafike i animacija drugih internet aplikacija; instalira se na internet preglednik kao dodatak). *TopBraid Ensemle* aplikacije bazirane su na *Flash* tehnologiji čime se osigurava dosljednost sučelja i izgleda na svim internet preglednicima.

Kada se sučelje izgradi kako bi bilo dostupno svim korisnicima, potrebno ga je postaviti na server koji pokreće još jedan proizvod *TopBraid* paketa, *TopBrid Live Enterprise Server*. *TopBraid Ensemble* je baziran na *MVC* (*model view controller*) arhitekturi koja omogćuje programerima jednostavno i brzo podešavanje aplikacije bazirane na željenom modelu. Aplikacija se jednostavno izrađuje postavljanjem prethodno konfiguriranih komponenti grafičkog sučelja za korisnike kao što su prozori s pretraživačima i prikaz na principu hijerarhijskog stabla te drugih komponenti koje omogućavaju korisniku pregled i uređivaje podataka.

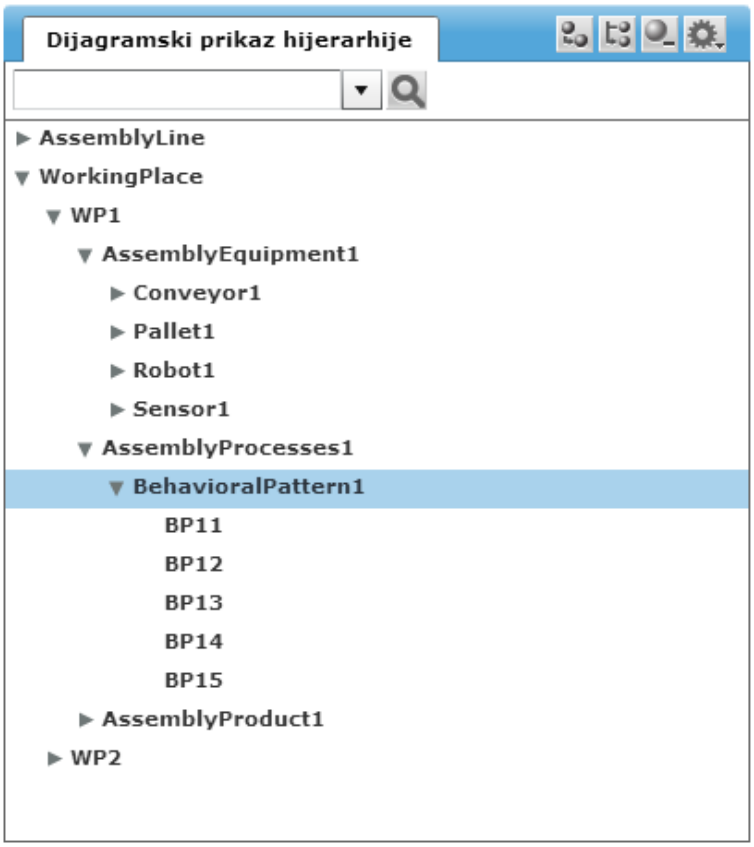
Za pokretanje *TopBraid Ensemblea* nije potrebna posebna dodatna instalacija osim prethodno instaliranog i pokrenutog *TopBraid Composer-Maestro Edition (Personal Server,* server na osobnm računalu-*local host)* ili *TopBraid Live (Enterprise Server)* alata. Kao što je već navedeno, iz financijskih razloga, za izradu ontologije je odabrana i instalirana besplatna probna verzija alata *TopBraid Composer-Maestro Edition* (i cjelokupni *TopBraid Maestro Edition* paket) s dozvolom uporabe od 30 dana. Instalirani paket *TopBraid Maestro Edition* sadrži *TopBraid Composer*, *TopBraid Ensemble* i probnu verziju *TopBraid Live* alata s osobnim serverom (*local host*). Koristeći probnu verziju alata, aplikacija je ograničena za korištenje na samo jednom računalu, odnosno osobnom serveru (*local host server*), budući da standadna verzija alata *TopBraid Live Enterprise Server* nije dostupna u probnoj verziji paketa.

6.2. Bitne komponente korisničkog sučelja

Sučelje se formira postavljanjem raznih unaprijed definiranih komponenti za željenu aplikaciju. Komponente sučelja se mogu međusobno povezati pomoću uzročnih događaja koji pri aktivaciji pokreću jednu od postavljenih komponenti. Bitne komponente kod korisničkog sučelja su stabla za prikaz hijerarhije, prozora za upisivanje *SPARQL* pita, prozora za ispis traženih rezultata, tražilicu (pretraga pomoću pojmova, a ne pomoću veza između trojki unutar ontologija), pregled podataka odabranih elemenata, klasa ili instanci. Postoji još niz komponenti od kojih su neke implantirane u izrađenu aplikaciju iz razloga da se prikažu sve mogućnosti *TopBraid Ensamble* alata i mogućnosti same aplikacije. Sve navedene komponente ključne su pri korištenju aplikacije kojoj je cilj olakšati korisnicima eksploataciju izrađenog ontološkog modela.

6.2.1 Dijagramski prikaz hijerarhije

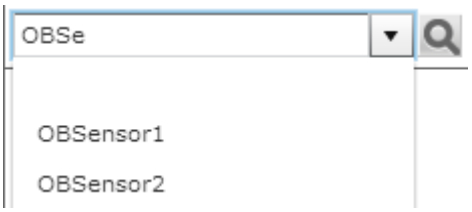
Komponenta za dijagramski prikaz hijerarhije [Slika 42.] prikazuje podatke prikazane u obliku stabla. Komponeta se unutar aplikacije originalno zove *Tree* (stablo). Korisnik po vlastitoj volji može proširivati ili smanjivati pregled podataka otvaranjem ili zatvaranjem grana dijagramskog prikaza hijerarhije. Otvaranjem jedne grane zapravo se dobiva uvid u podklase koje sadrži odabrana klasa. Pregled podklasi, otvaranje ali i zatvaranje grana vrši se klikom miša na trokutić pored odabrane klase. Ukoliko je trokut usmjeren vodoravno (jednim kutem prema desno), tada je klasa zatvorena i njezine podklase nisu vidljive. Pritiskom na trokut pojavljuju se podklase odabrane klase, a trokut pri tome gleda prema dolje što znači da je grana klase otvorena.



Slika 42. Dijagramski prikaz hijerarhija (stablo, tree)

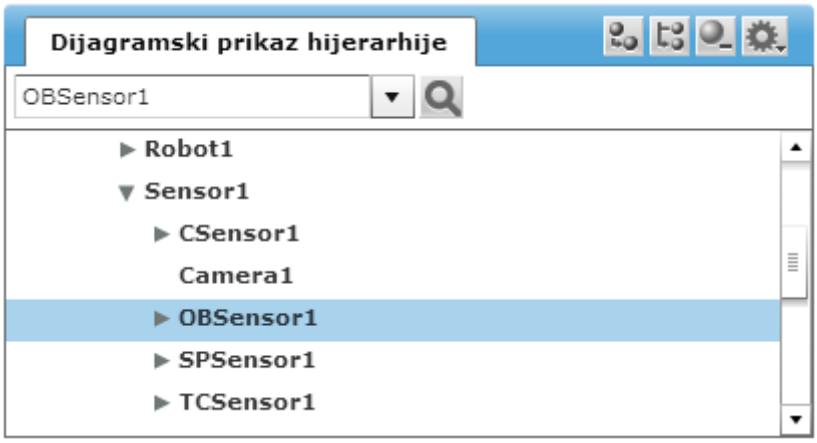
Ukoliko je dopušteno (pomoću prvotnih postavki), dijagramskom prikazu se mogu dodavati ali i brisati grane, odnosno klase. Dodaci za dodavanje i brisanje klasa su vidljivi na slici [Slika 42.] u gornjem desnom kutu.

U većini slučajeva nema potrebe za takvom opcijom jer su ontologije unaprijed izrađene pomoću *TopBraid Composera,* no ako se želi ontologiji pristupiti po sloganu na kojem se bazira semantički *Web,* a to je: *Anyone can say anything about any topic (AAA*- bilo tko može reći/napisati bilo što o bilo kojoj temi), u tom je slučaju poželjno pri izradi aplikacije unutar sučelja postaviti opciju za dodavanje i brisanje klasa. Unutar aplikacije za hijerarhijski prikaz podataka moguće je i integrirati tražilicu pojmova (*Search Form*) koja je vrlo korisna pri snalažanju kod velikih i opširnih ontologija.



Slika 43. Tražilica unutar aplikacije za dijagramskog prikaza hijerarhije

Upisivanjem pojmova tražilica automatski nadopunjuje pojmove prema bazi podataka koju sadrži [Slika 43.] Pretraživanjem traženog pojma, grana u kojoj se pojam nalazi otvori se, a pojam se označi (u našem primjeru svijetlo plavom bojom [Slika 44.]).



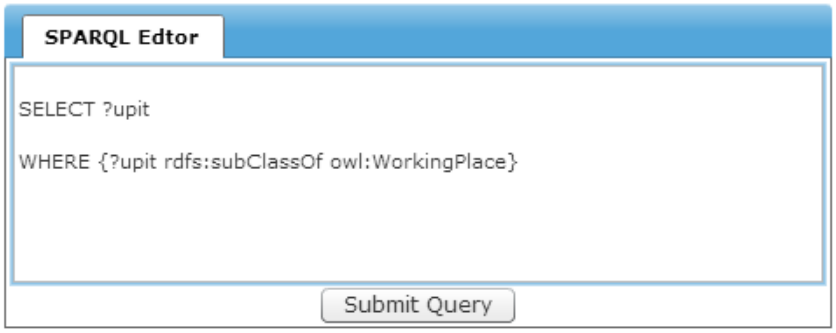
Slika 44. Označavanje traženog pojma pomoću integrirane tražilice

Prikaz hijerarhijskog stabla je identično (uz neke estetske promjene) prikazu koji se dobije pomoću *TopBraid Composer* alata. Prema tome, dijagramski prikaz hijerarhije prikazuje međusobni odnos klasa (podklasa, superklasa, sestrinska klasa).

Navedeni način prikaza je općenito najzanimljiviji jer direktno pokazuje međusobni odnos klasa te se dobiva jasnija slika o sustavu koji se želi opisati. U prikazanom primjeru hijerarhija je prikazana bez početne superklase *Thing* koja je neizostavni element ontologije (svaka klasa ontologije je podklasa klase *Thing* koja je prema tome superklasa svim klasama ontologije). Superklasa *Thing* se može a i ne mora prikazati u dotičnom prikazu, ovisi o želji programera. Uključivanje i isključivanje prikaza superklase vrši se u postavkama dijagrama postavljanjem vrijednosti *true* ili *false* na opciju: *Show the root of the tree in the display*. Izrada sučelja pomoću *TopBraid Ensemble* alata dopušta i mogućnost postavljanja bilo koje klase kao superklasu sustava, ukoliko se želi promatrati ili prikriti samo jedan dio ontologije.

6.2.2. SPARQL Editor

SPARQL Editor je komponenta sučelja koja služi za unos *SPARQL* upita. Nakon što se unese točni željeni upit pritiskom na tipku *Submit Query* izvršava se *SPARQL* upit čija rješenja dobijemo u posebnom prozoru koji je objašnjen u sljedećem poglavlju.



Slika 45. Komponenta za unos SPARQL upita

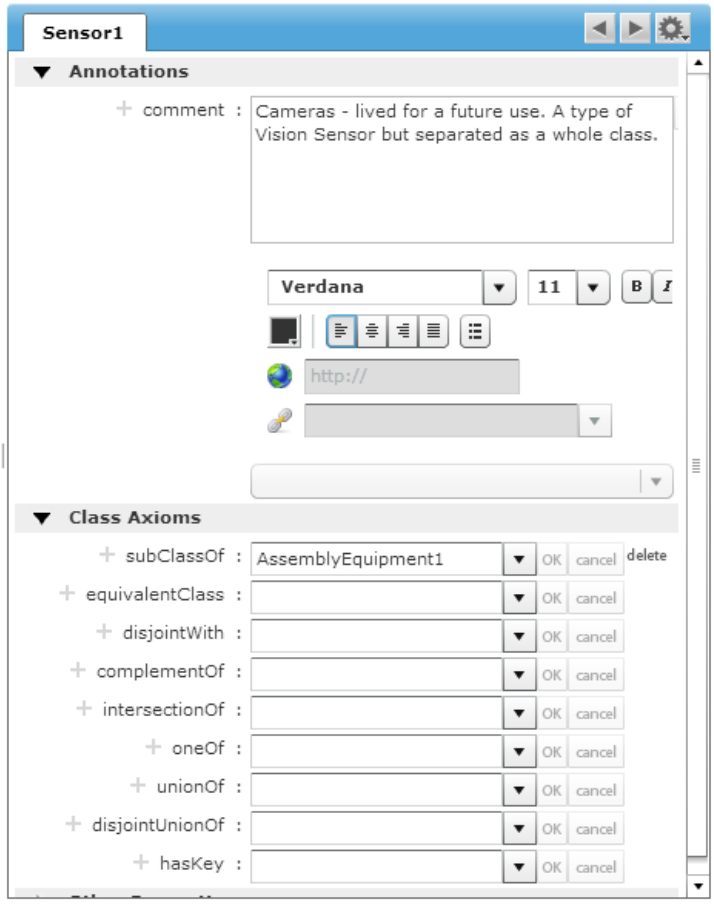
Princip unosa *SPARQL* upita je identična kao i kod *TopBraid Composer* alata, unosi se pomoću trojki. Aplikacija koristi isti *SPARQL engine* kao i *TopBraid Composer*. Pri uređivanju postavki komponente *SPARQL Editor* moguće je postaviti poveznice s ostalim komponentama koje također služe za pretraživanje ontologije. Takve komponente su *Search Form* i *Visual Query Editor*. Prozor za pretraživanje je moguće proširiti ili smanjiti po želji. Nakon upisanog upita moguće ga je sačuvati te ponovno iskoristiti pri sljedećoj primjeni.

6.2.3 Prikaz rezultata i ostalih informacija

Pomoću komponente *Results Grid* (u izrađenom sučelju komponenta je preimenovana u „Rezultati pretraživanja“) prikazuju se setovi rezultata koji se dobiveni pomoću dostupnih pretraživača. Rezultat se može dobiti jednostavnim izvršenjem *SPARQL* upita, pretraživanjem dijagrama za prikaz hijerarhije ili preko *Graph Editora*. Dodatni prikaz podataka dobivenih rezultata može se dobiti klikom miša na rezultat pretrage. Komponenta ima mogućnost prikaza rezultata po početnim slovima ili ukupni prikaz rezultata. Pri odabiru prikaza po početnim slovima, iznad rezultata označen je broj prikazanih rezultata od ukupnog broja.

Slika 46. Komponenta Result Grid

Po želji se komponenti mogu instalirati dodaci koji omogućuju upravljanje rezutatima, njihovo pohranjivanje, brisanje i eksportiranje u drugačijim formatima. Vidljivo je da se pomoću navedene komponente dobivaju rezultati pretrage, dok nam podaci o rezultatu nisu dostupni. Za prikaz svojstava rezultata postavljena je komponenta *Form* (obrazac) koja nam prikazuje sve dostupne podatke o odabranom izvoru. Prikaz svojstava klase *Sensor1* vidljiv je na slici [Slika 47.]. Komponenta *Form* može se predstaviti kao *TopBraid Ensemble* verzija centralnog prozora za unos podataka kao ontologija u *TopBraid Composer* alatu. Jednostavnim unosom postavki komponenata nam omogućuje prikaz informacija o određenoj klasi, instanci ili svojstvu zavisno o postavljenoj opciji *Displayed attribute*.



Slika 47. Komponenta Form za prikaz svojstava odabranog izvora (edit)

Postavljena vrijednost komponente *Form* je *review* dok je na slici [Slika 47.] postavljena na vrijednost *edit* (uređivanje). Treća opcija je postavljanje vrijednosti na *view*, čime je omogućen samo pregled bez mogućnosti izmjene. Izgled je isti, samo se prelaskom miša više ne pojavljuje opcija *edit* pomoću koje se može izmjeniti sadržaj. Pomoću opcije *edit* možemo postaviti da je izvor informacija svima dostupan za izmjenjivanje i nadopunjavanje. Uz nadopunjavanje komentara, upisivanje novih svojstava, moguće je izmjenivati oblik i boju upisanog teksta. Komponenta sadrži i navigacijske strelice kojima se lakše možemo vratiti na već pregledani izvor kako ne bi bili primorani ponovno pretraživati ontologiju. Vrijednosti na obrazcu su hipertekstualne poveznice koje povezuju druge obrazce za prikaz podataka određenih izvora. Za podatak prefiksa *xsd:* (gdje se nakon prefiksa nalazi neka URL adresa), vrijednost može poslužiti kao poveznica s nekom od web stranica ili dokumentom drugačijeg formata. Podatak koji je prikazan pomoću komponente je vrijednost trenutno odabranog izvora gdje je vrijednost definirana *TopBraid Suite* riječnikom (*TBS dictionary*).

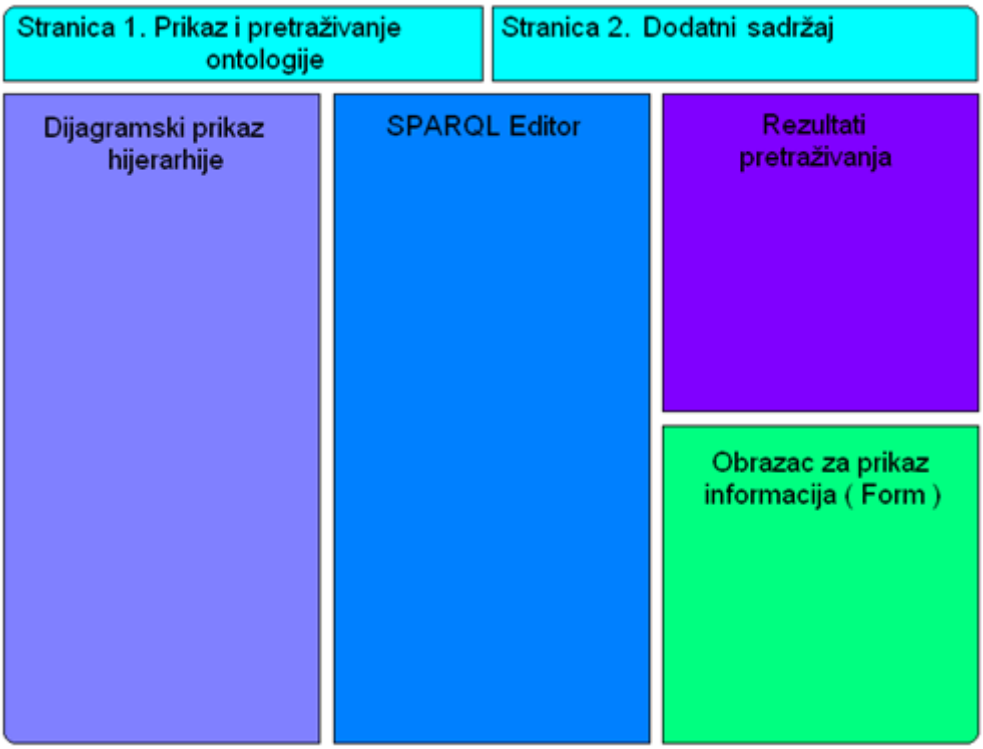
6.3. Izrada sučelja TopBraid Ensamble alatom

Za izradu sučelja koristi se *TopBraid Composer Maestro Edition* koristeći server na osobnom računalu (*personal host*). Sučelje se izrađuje tako da se u aplikaciju unose već gotove nedefinirane koponente koje, da bi međusobno djelovale, moraju biti povezane određenim funkcijama. Povezivanje komponenti funkcijama vrši se pomoću opcija *post* i *listen*. Funkcije mogu imati proizvoljna imena. U jednom od naših slučajeva koristili smo naziv *Tree click*. Navedena funkcija služi da se odabirom klase u dijagramskom prikazu hijerarhije njezini podaci prikažu na komponentama za prikaz informacija. Pri tome je kod dijagramskog prikaza hijerarhije funkcija *Tree click* postavljena pod *post* opcijom, a kod komponenti za prikaz informacija pod *listen* opcijom.

Slika 48. Proces izrade korisničkog sučelja

Dijagram cijelokupnog procesa izrade korisničkog sučelja vidljiv je na slici [Slika 48.]. Proces izrade korisničkog sučelja započinje se izradom ontološkog modela, što je već otprije napravljeno. Zatim je potrebno skicirati izgled sučelja, kako bi se unaprijed znalo kako i gdje smjestiti koju komponentu. Pri tome treba paziti na funkcionalnost sučelja, nepotrebno je postavljati komponente koje se neće koristiti, time bi se samo nepotrebno opterećivalo sučelje. Skica izgleda sučelja, odnosno smještaja komponenti je vidljiva na slici [Slika 49.] Nastojalo se postaviti osnovne komponente za pregled i pretraživanje ontološkog modela tako da budu dostupne sve moguće informacije o željenoj ontologiji. Nakon skiciranja sučelja, potrebno je započeti izradu kreiranjem nove aplikacije, te odabirom izgleda ključnih stranica i ključnih komponenti.

Nakon odabira komponenti potrebno ih je određenim funkcijama međusobno povezati i konfigurirati. Konfiguracija se vrši posebnim opcijama, a povezivanje otprije spomenutim funkcijama pomoću *post* i *listen* opcija. Nakon što je sučelje izrađeno, potrebno ga je testirati. Testiranje se može vršiti postavljanjem *SPARQL* upita ili pomoću tražilice postavljene unutar dijagramskog prikaza hijerarhije.

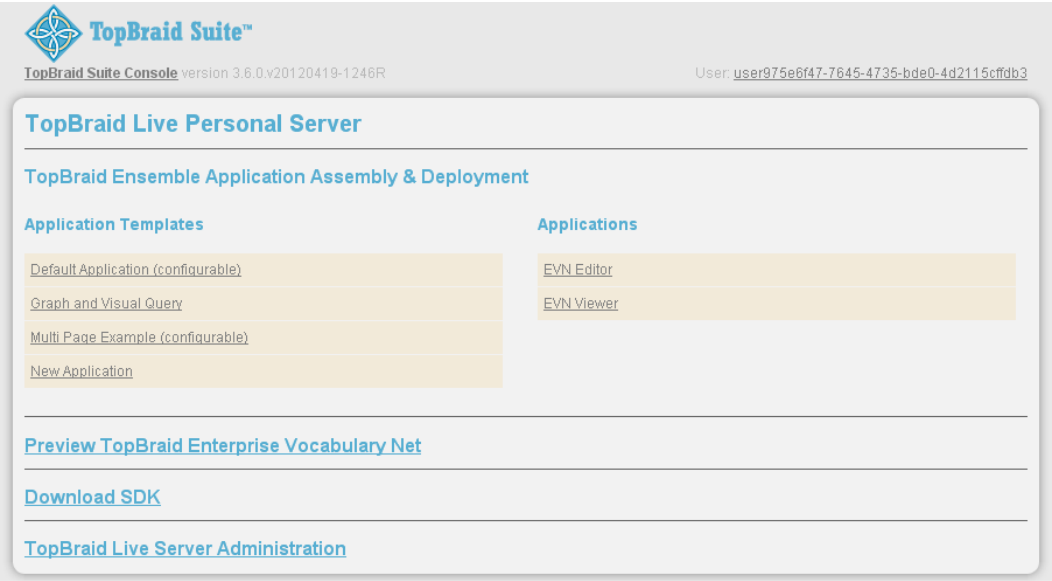


Slika 49. Skicirani izgled korisničkog sučelja

Za izradu nove aplikacije potrebno je pokrenuti osobni server na računalu. Za pokretanje osobnog servera u internet pregledniku potrebno je unijeti sljedeću URL adresu: *http://localhost:8083/tbl.* Nakon što smo unijeli potrebnu adresu, u interet pregledniku nam se otvora *TopBraid Suite* konzola s *TopBraid Live Persoal Server*-om vidljiv na slici [Slika 50.]. Preko konzole su nam dostupne neke od opcija od kojih su nama najvažnije *Application Templates* i *Applications. Applications Templates* nam daje mogućnost odabira prgramski postavljenih obrazaca sučelja, dok nam *Applications* daje mogućnost odabira obrazaca sučelja koje smo sami otprije napravili te ih sačuvali za ponovno korištenje. U konzoli *TopBraid Live Personal Server* nalazi se još jedna bitnija opcija, a to je *TopBraid Live Server Administartion*. Pomoću opcije *TopBraid Live Server Administration* moguće je mijenjati neke od postavki osobnog servera i vidjeti informacije koje se tiču dotičnog osobnog servera.

Također je moguće i postaviti zaštitu kojom se onemogućava neovlašteni pristup osobnom serveru. Moguće je upravljanje rječnikom pojmova servera i postavljanje nekih novih komponenti u sučelje, odnosno nadogradnja sučelja ukoliko se koristi stariji model.

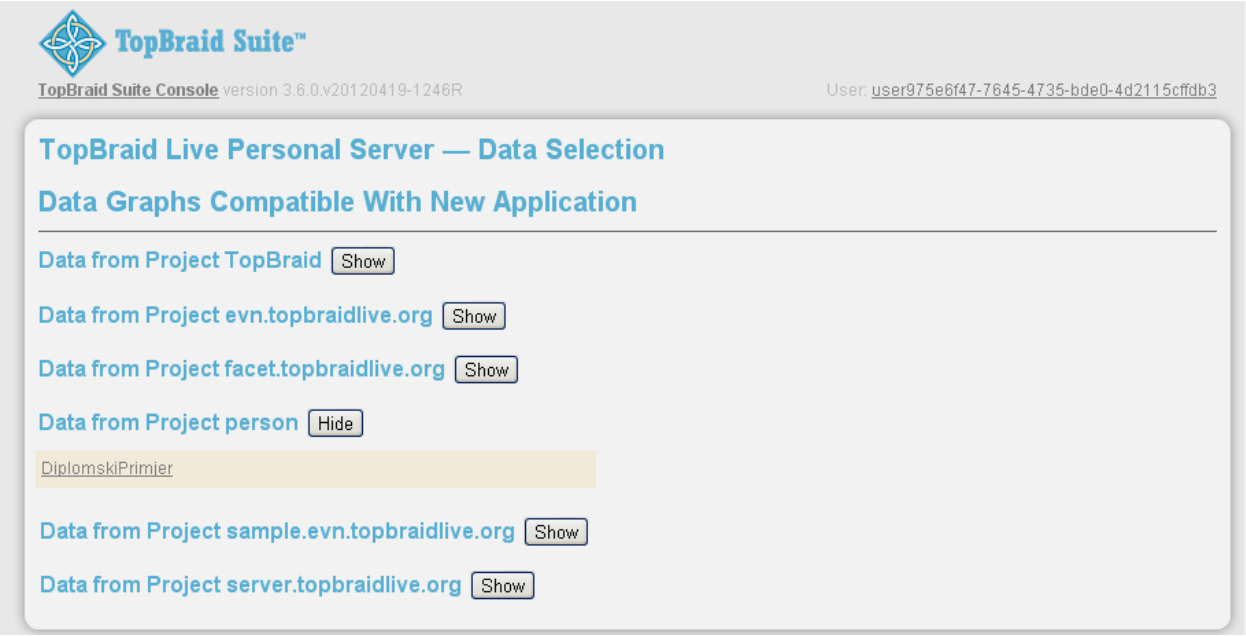
Kod izrade korisničkog sučelja pri pokretanju osobnog servera potrebno je imati pokrenut *TopBraid Composer* u kojem je ukomponiran alat *TopBraid Ensemble*. Pri izradi ovog sučelja korišetna je verzija 3.6.0. *TopBraid Composer Maestro Edition* koja nije najnovija verzija paketa. Razlog tome je što u novijim verzijama (4.2.0.- verzija u kojoj je rađen ontološki model) nije omogućeno korištenje *TopBraid Ensemble* alata, već se koriste neki drugi alati (*SWP- SPARQL Web Pages*, više o tom alatu u sljedećem poglavlju). Smatra se da će nova verzija *TopBraid Ensemble* alata za *TopBraid Composer* 4.2.0. biti dostupna krajem 2013. godine. Verzija 3.6.0. je potpuno kompatibilna s 4.2.0. verzijom *TopBraid Composer* alata te se ontološki model napravljen u jednoj verziji bez problema može koristiti u drugoj.



Slika 50. TopBraid Suite konzola

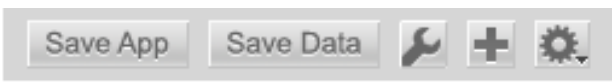
Izrada sučelja započinje odabirom opcije *New Application*. Odabirom navedene opcije započinje se izrada sučelja od samog početka. Ukoliko se želi preraditi jedan o standardnijih oblika sučelja, može se odabrati jedno od tri ponuđena sučelja koja sadrže osnovne komponente i jednostavnog su izgleda. Nakon odabira željene opcije, potrebno je odabrati ontološki model za koji želimo napraviti sučelje.

Jednom napravljeno sučelje se može koristiti na neograničen broj ontoloških modela bez posebne pripreme, pri tome se sučelje prilagođava odabranom ontološkom modelu. Na ponuđenim mapama potrebno je odabrati onu koja sadrži željeni ontološki model pritiskom na tipku *Show* te odabirom modela [Slika 51.].



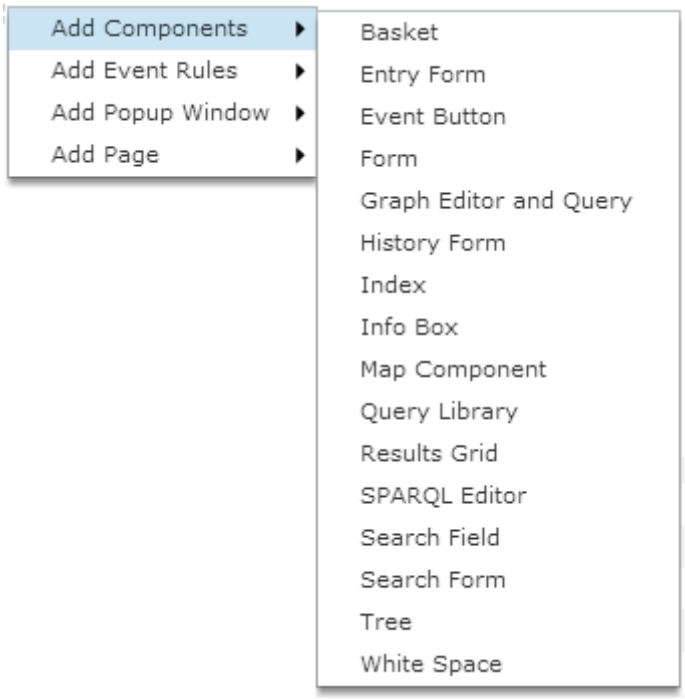
Slika 51. TopBraid Live Personal Server – odabir ontološkog modela

Nakon odabira željenog ontološkog modela, otvara nam se prozor nove aplikacije s ponuđenom alatnom trakom koju je također moguće uređivati. U alatnoj traci nalaze se tipke za sačuvanje aplikacije i podataka promjenjenih unutar *TopBraid Ensemble* alata i prikaz postavki odabrane stavke. Ukoliko se te tipke maknu, neovisno o promjenama podataka, te promjene se neće moći sačuvati i pri ponovnom pokretanju aplikacije podaci će biti postavljeni po originalnim postavkama. Bitne tipke za izradu sučelja su tipka za prikaz konfiguracije (ključ) i tipka za dodavanje komponenti (plus) [Slika 52.].



Slika 52. Alatna traka za izradu sučelja

Pritiskom na tipku za dodavanje novih komponenti otvara nam se početni prozor s ponuđenim opcijama za dodavanje. Prema slici [Slika 53.] vidljivo je da je moguće dodati nove komponente (*Add Components*), pravila za aktiviranje komponenti (*Add Event Rules*), iskočnih prozora (*Add Popup Window*) i dodatnih stranica za postavljanje komponenti (*Add Page*).



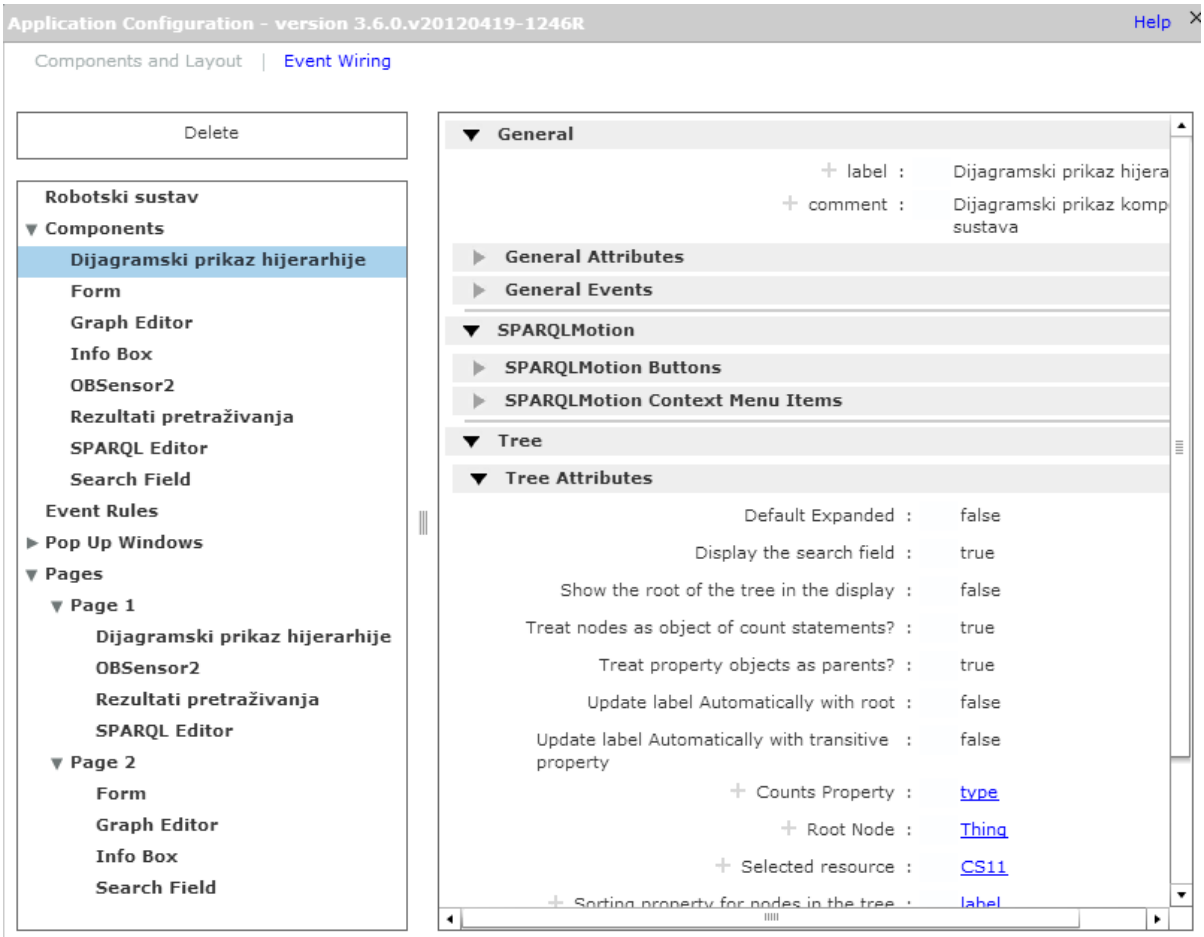
Slika 53. Opcije za dodavanje novih komponenti

Odabirom dodavanja novih komponenti otvara nam se mogućnost postavljanja velikog broja komponenti od koji su izabrane: *Tree, SPARQL Editor, Results Grid i Form*. Funkcije i sadržaji navedenih komponenti opisani su u prethodnom potpoglavlju. Uz navedene osnovne komponente koje su bitne za funkcioniranje željenog sučelja, za prikaz su na drugoj stranici postavljene i komponente *Event Button, Info Box* i *Graph Editor and Query*.

Korištenje sučelja na više stranica je korisno jer omogućuje da se informacija distribuira preko više stranica i time izbjegne prikaz previše informacija na jednoj stranici. Broj stranica i stupanj detalja prikazan na svakoj stranici zavisi o svrsi sučelja prema kojoj će se izrađivati struktura sučelja. Za prezentirano sučelje odabrane su dvije stranice. Na prvoj stranici nalaze se bitne komponente za pregled i pretraživanje ontološkog modela, a na drugoj su dodatne komponente koje se mogu a i ne moraju koristiti, postavljene su jedino iz razloga da se pokaže način njihovog postavljanja i rada.

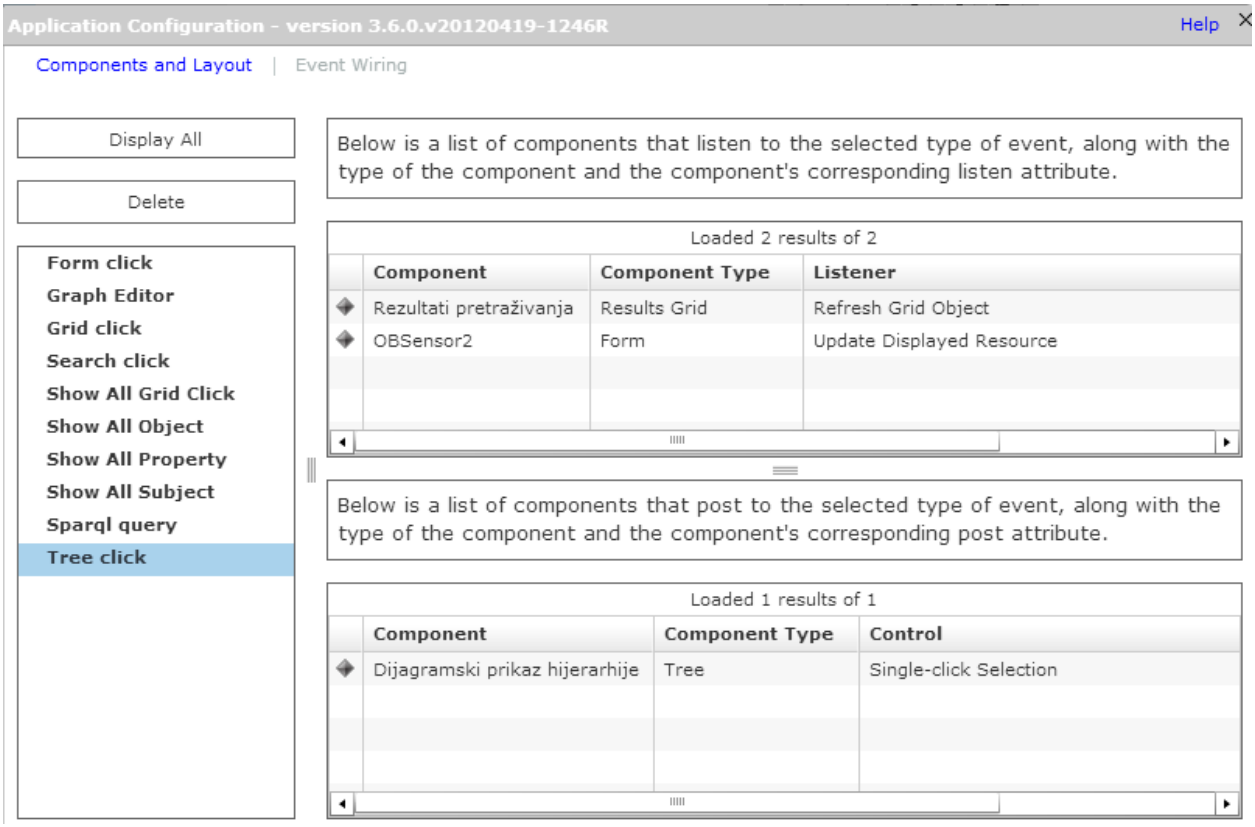
Dodavanje nove stranice vrši se pomoću opcije *Add Page*. Vrlo bitna stavka pri izradi višestraničnog sučelja je postavljanje opcije *Multi-paged Application* (nalazi se u prozoru za konfiguraciju aplikacije) na vrijednost *true*. Ukoliko je vrijednost navedene opcije *false*, neće biti moguće dodavanje dodatnih stranica.

Za početak postavljanja komponenti prvo je odabrana komponenta *Tree* te preimenovana u Dijagramski prikaz hijerarhije. Njoj je pomoću aplikacijske konfiguracije [Slika 54.] dodana tražilica pojmova unutar dijagrama. Na isti način dodane su i komponente *SPARQL Editor*, *Results Grid* uz preimenovanje u Rezultati pretraživanja i *Form* čije ime ovisi o stavci čije informacije prikazuje. Prostorno postavljanje komponenti je jednostavno i vrši se pomicanjem pomoću miša, moguće je i mijenjati dimenzije komponenti. Ukoliko se neka komponenta poželi ukloniti iz aplikacije, to se vrši pomoću tipke *Delete* (brisanje) koja se nalazi u prozoru za konfiguraciju aplikacije.



Slika 54. Prozor za konfiguraciju aplikacije

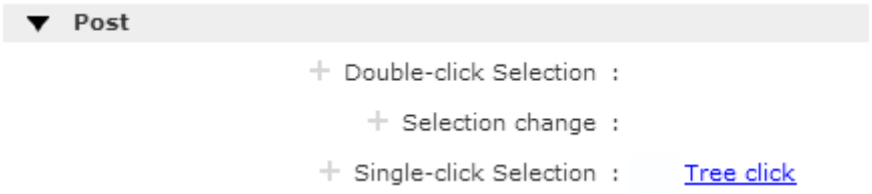
Sljedeći korak je povezivanje postavljenih komponenti pomoću raznih proizvoljno određenih funkcija. Funkcije omogućuju međusobnu komunikaciju između komponenti. Komunikacija se obavlja spomenutim opcijama *post* (objava ili pozivanje) i *listen* (slušanje). Funkcije se odrađuju klikom miša na određenu stavku. Kada se izvrši odabir neke stavke klikom miša istovremeno se vrši „objava" neke funkcije dok druge komponente „slušaju" tu objavu. Ukoliko komponenta ima zadanu postavku za prihvaćanje te objave, ona tada izvršava zadani postupak. Imena funkcija se dodjeljuju proizvoljno i njihove poveznice je moguće pregledati u prozoru za konfiguraciju aplikacija pod *Event Wiring* [Slika 55.].



Slika 55. Prozor za pregled izrađenih funkcija Event Wiring

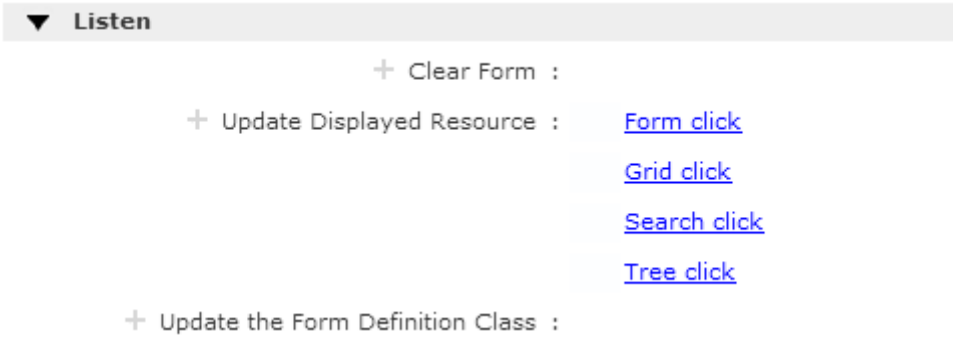
Izrada funkcije za prikaz informacija u obrascu (*Form*) neke stavke koje smo odabrali u dijagramskom prikazu hijerarhije započinje odabirom imena funkcija. U standardno postavljenim sučeljima (korištenjem neke od *Default Application*) funkcija koja je vezana uz dijagram naziva se *Tree click*, zbog jednostavnosti izrađenoj funkciji dan je identičan naziv. Pod opcijom *Event* (događaji) nalaze se već spomenute opcije *Post* i *Listen.*

Kod dijagramskog prikaza hijerarhije je pod opcijom *Single-click Selection* (odabir jednim klikom miša) postavljena odabrana funkcija *Tree click* [Slika 56.]. Time je postavljeno objavljivanje događaja.



Slika 56. Postavljanje funkcije Tree click kod dijagramskog prikaza hijerarhije (Tree)

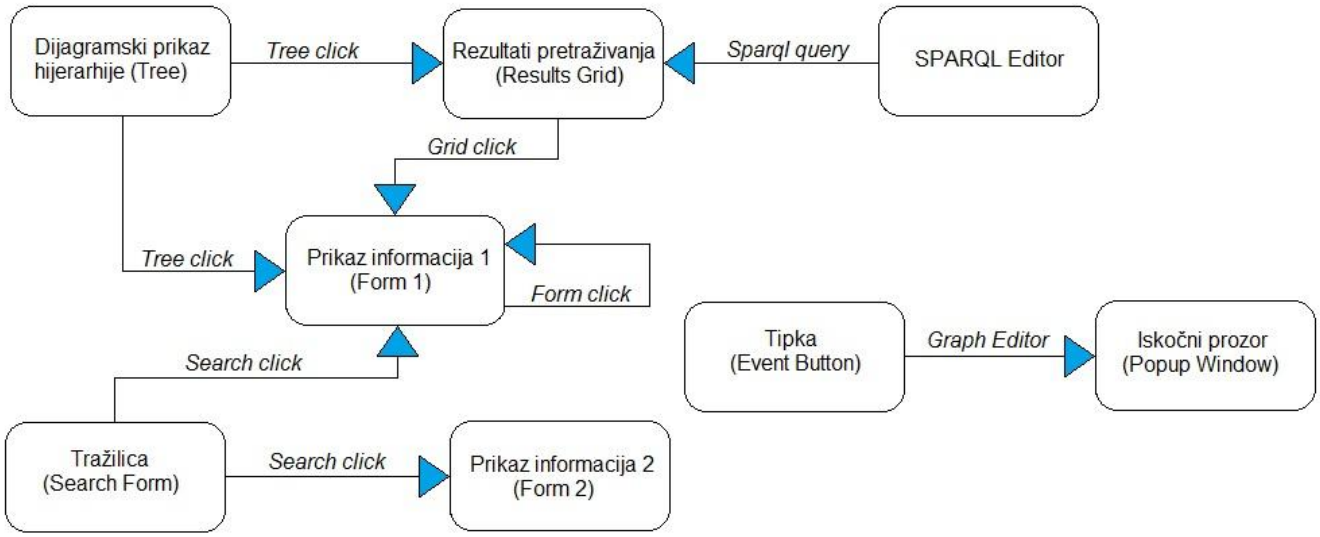
Nakon postavljanja objavljivanja događaja, postavlja se tzv. „slušanje“ (praćenje) događaja koje se izvodi tako da se funkciju *Tree click* (mogu se unijeti i druge funkcije ako se želi da se prikažu u obrazcu) upiše pod opcijom *Update Displayed Resource* [Slika 57.]. Ovim postupkom uređena je funkcija *Tree click* te klikom na bilo koju stavku u dijagramskom prikazu hijerarhije dobit ćemo njezine podatke u obrascu.



Slika 57. Postavljaje funkcije Tree click kod obrazca za prikaz informacija (Form)

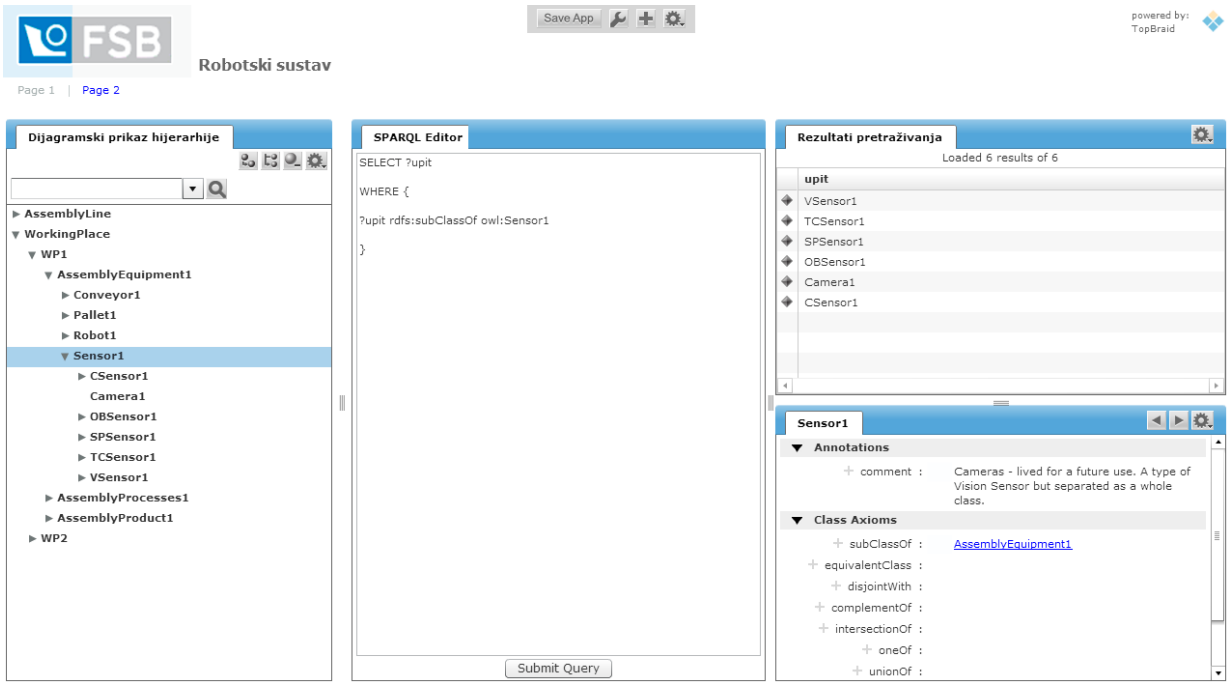
Na sličnom principu zasniva se i funkcioniranje dodanih tipki za pozivanje iskočnih prozora. Iskočni prozori dodaju se odairom opcije *Add Popup Window,* a tipke na odabiru *Event Button* kod opcija za dodavanje novih komponenti. Dodani iskočni prozor se povezuje s tipkom također pomoću funkcija. Iskočni prozor je taj koji „sluša“ pozivanje tipke klikom miša. Kod iskočnog prozora pod opcijom *Open the popup* je postavljena funkcija *Graph Editor*, što znači da se iskočni prozor otvara pozivanjem dotične funkcije. Kod tipke funkcija je postavljena pod opcijom *Click Event*.

Veličina i naziv iskočnog prozora lako su promjenjivi, a prozor se zatvara pritiskom na ikonu *X* u gornjem desnom kutu. Primjer funkcioniranja tipke i iskočnog prozora prezentiran je na drugoj stranici korisničkog sučelja. Na slici [Slika 58.] može se vidjeti korištene funkcije pri pozivanju rezultata i iskočnog prozora.

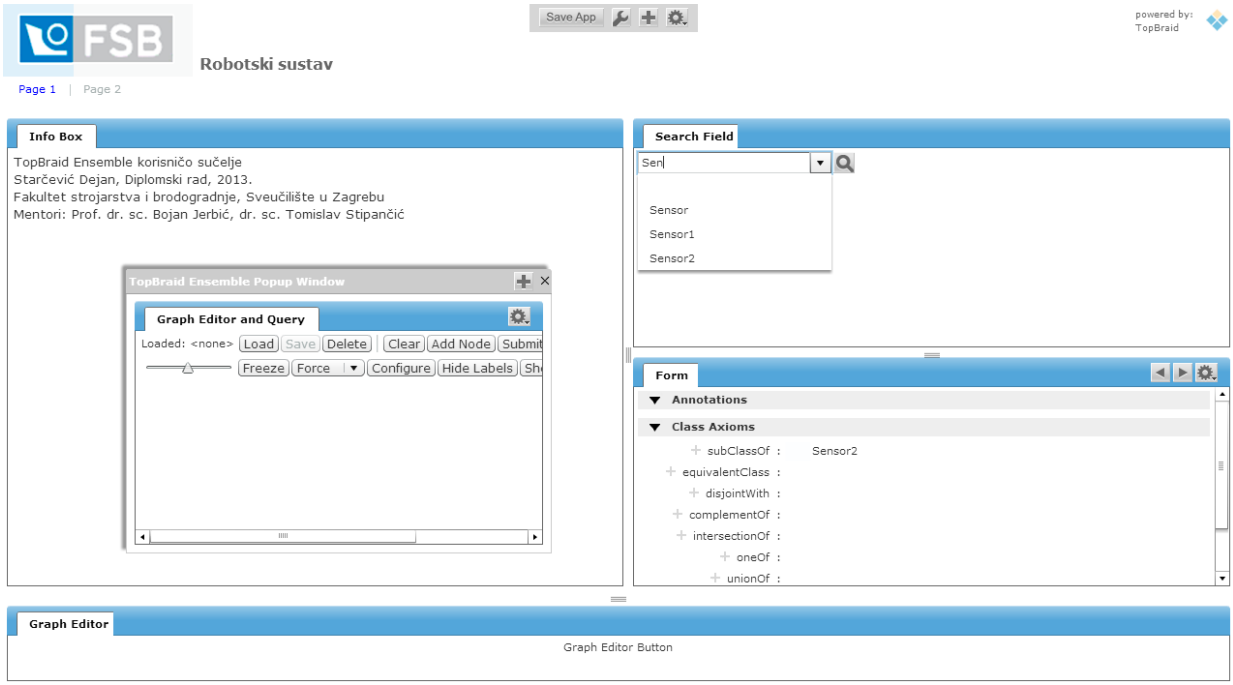


Slika 58. Funkcije i događaji korisničkog sučelja

Testiranje gotove aplikacije vrši se jednostavnim isprobavanjem rada njezinih komponenti. Aplikacija se po završetku sačuva pod imenom *AplikacijaDiplomski.ttl* te je spremna za ponovno pokretanje i korištenje. Pri upisu *SPARQL* koda u *SPARQL Editor* te pokretanjem *SPARQL* upita dobiva se rezultat upita pomoću komponente *Results Grid* (Rezultati pretraživanja). Klikom miša na jedan od dobivenih rezultata dobiva se prikaz detalja izabrane stavke u obrazcu za prikaz informaija (*Form*). Detalji o određenoj stavci se također prikazuju klikom miša na jednu od njih u Dijagramskom prikazu hijerarhije. Na drugoj stranici sučelja, kliom na tipku *Graph Editor Button* pojavljuje se iskočni prozor s komponentom *Graph and Query Editor.* Upisivanjem i pretraživanjem pojmova dobiva se rezultat pomoću obje komponente *Form.* Prema navedenim doađajima koji su bili predviđeni može se zaključiti da aplikacija djeluje bez greške te da je uspješno izvedena. Na slikama [Slika 59.] i [Slika 60.] može se vidjeti izgled gotovog sučelja.



Slika 59. Prva stranica gotovog korisničkog sučelja



Slika 60. Druga stranica gotovog korisničkog sučelja

7. WEB APLIKACIJA

Da bi se korisničko sučelje izrađeno *TopBraid Ensemble* alatom postavilo u funkciju (korištenje putem weba, a ne samo na lokalnom računalu putem *TopBraid Live* lokalnog severa) potrebno je imati i *TopBraid Enterprise Server*. Navedeni alat se, kao i svi alati unutar *TopBraid* paketa, naplaćuje te njegova primjena u ovom trenutku iz financijskih razloga nije moguća. Stoga se krenulo u traženje drugog rješenja za prezentaciju i pretraživanje *RDF* trojki putem weba.

Uzmajući u obzir da aplikacija za korištenje treba biti jedostavna te da njezina izrada nije financijsko opterećenje, odabrano je rješenje izrade web stranice koja će sadržavati bazu *RDF* podataka i mogućnost pregleda i pretraživanja tih podataka putem *SPARQL* upita. Web stranica je izrađena na temelju *ARC2 libraryja* za *PHP* i *MySQL* koji su besplatni. *ARC2* je pokrenut kao jednostavni *RDF* sustav za rasčlanjivanje (*parsing*) i serijalizaciju *RDF/XML* datoteka. Kasnije se razvio u kompletni okvir s mogućnošću pohrane i pretraživanja podataka. Trenuto je zbog svoje jednostavnosti i pristupačnosti jedan od najkorištenijih *RDF libraryja*. Korištenjem *ARC2*-a, *PHP*-a i *MySQL*-a moguće je izraditi vrlo kvalitetnu i korisnu web aplikaciju (semantičku web aplikaciju) i *RDF* baze podataka.

7.1. Korištene tehnologije

Izrađena web aplikacija se postavlja na server i time je dostupna putem weba. Pri izradi i za demonstraciju korišten je *WAMPserver*. *WAMPserver* (*Windows, Apache, MySQL, PHP/Perl/Python server*) je miniserver koji se može pokrenuti s bilo kojeg *Windows* operativnog sustava. *WAMP* predstavlja nezavisno izrađene programe *Apache 2*, *PHP 5* i *MySQL*, sadrži i *phpMyAdmin* i *SQLitemanager* koji pružaju grafičko korisničko sučelje za upravljanje postavljanom bazom podataka. *WAMPserver* preuzet je s web stranice: *http://www.wampserver.com*. Instalacija je jednostavna i vrši se automatski, a korištenje je moguće odmah nakon instalacije i bez mijenjanja originalnih postavki. Datotekama, web stranicama koje se stavljaju i poslužuju pomoću *WAMP* servera može se pristupiti upisivanjem adrese *http://localhost* ili *http://127.0.0.1* u svoj web preglednik. Za pristup navedenim adresama *WAMP* server mora biti upaljen i pokrenut. *Apache HTTP server*, često se naziva samo *Apache*, je software program poznat po tome što je imao veliku ulogu u razvoju *World Wide Weba*.

Apache je prvotno dizajniran za *Unix* okruženje, kasnije je prebačen na *Windowse* i druge operativne sustave. Apache web server pruža razne web server značajke i aplikacije uključujući *CGI, SSL* i virtualne domene. Software *Apache* je besplatan, a u svrhu izrade web aplikacije instaliran je zjedno s *WAMP* serverom. *Apache Jena* koristi se za izradu semantičkih aplikacija. *Apache Jena* je okvir koji daje podršku za *RDF* raščlanjivanje, spremanje i pretraživanje.

PHP (PHP Hypertext Preprocessor) je programski jezik raširen i popularan pri izradi web stranica, ali i programski jezik za opću namjenu. *PHP* je namijenjen za programiranje aplikacija koje se koriste na serveru. U trenutku kad neki korisnik posjeti jednu od *PHP* stranica, web server automatski obrađuje *PHP* kod na osnovu kojeg određuje što će prikazati korisniku. Sve ostalo kao npr. matematičke operacije, operacije s datotekama, varijable itd. ne prikazuje korisniku, te korisnik pomoću web preglednika prima generiranu *HTML* stranicu bez PHP koda. Većina sintakse je preuzeta iz *C, Jave i Perla* s nekoliko svojih specifičnosti. Bitno pri korištenu *PHP* skripti ili stranica je da web server podržava *PHP*, osim nekih besplatnih, većina komercijalnih servera podržava *PHP*.

MySQL (*My Structured Query Language*) je besplatan, *open source* sustav za upravljanje bazom podataka. Jedan od čestih izbora baze za projekte otvorenog koda, te se distribuira kao sastavni dio serverskih *Linux* distribucija, no također postoje inačice i za ostale operativne sustave (*Mac OS, Windows*). *MySQL* baze su relacijskog tipa, koje su se pokazale kao najbolji način skladištenja i pretraživanja velikih količina podataka i u suštini predstavljaju osnovu svakog informacijskog sustava, tj. temelj svakog poslovnog subjekta koji svoje poslovanje bazira na dostupnosti kvalitetnih i brzih informacija. *MySQL* baza je slobodna za većinu uporaba. Ranije u svom razvoju, *MySQL* baza podataka suočila se s raznim protivnicima *MySQL* sustava organiziranja podataka jer su joj nedostajale neke osnovne funcije definirane *SQL* standardom. Naime, *MySQL* baza je optimizirana kako bi bila brza. Nasuprot tome, vrlo je stabilna i ima dobro dokumentirane module i ekstenzije te podršku od brojnih programskih jezika: *PHP, Java , Perl, Python.*

Za pristup *RDF* podacima koristiti će se *SPARQL endpoint. SPARQL endpoint* predstavlja protokol kojim možemo slati upite prema otvorenim *RDF* bazama podataka. Navedenom tehnologijom možemo pretraživati vlastitu bazu podataka kao i baze podataka nekih drugih baza i aplikacija. Najčešće su rezultati prikazani kao tablica, ali postoji i mogućnost prikaza rezultata putem nekih od načina serijalizacije.

7.2. Izrada web aplikacije

Nakon potrebnih instalacija i izrada potrebnih direktorija, potrebno je kreirati novu bazu podataka. Baza podataka izradit će se u aplikaciji *phpMyAdmin* koja je dio *WAMPservera*. Opcijom *Create database* kreira se baza željenog imena. Baza za spremanje *RDF* podataka u ovom radu nazvana je *sustavdipl*. Uz kreiranje baze podataka kreira se i korisničko ime i lozinka. Nakon kreiranja baze podataka, korisničkog imena i lozinke, te podatke je potrebno unijeti u ostale datoteke koje te podatke koriste, a to su *config.php, load.php i endpiont.php*. Dio koda koji uz navedene podatke sadrži i podatke o nazivu spremnika *RDF* trojki i poslužitelja (poslužitelj je na lokalnom računalu, *localhost*) prikazan je na slici [Slika 61.] Sve navedene datoteke u kojima je potrebno promijeniti podatke nalaze se u direktoriju *www\arc2-starter-pact*.

```
'db_host' => 'localhost',
'db_name' => 'sustavdipl',
'db_user' => 'Dejan',
'db_pwd' => 'diplomski',
'store_name' => 'sandbox',
```

Slika 61. Prikaz koda s podacima o bazi podataka (database information)

Kada se kreira baza za podatke *sustavdipl* i u tu bazu učitaju neki podaci, *ARC2* automatski izrađuje tablice za spremanje podataka iz kojih će se kasnije pretraživanjima izvlačiti. *ARC2* kreirane tablice vidljive su na slici [Slika 62.]. Najzanmljivija je tablica sandbox_triple u koju se spremaju *RDF* trojke.

Table
sandbox_g2t
sandbox_id2val
sandbox_o2val
sandbox_s2val
sandbox_setting
sandbox_triple
6 tables

Slika 62. Automatski kreirane tablice u bazi podataka

ARC2 se sastoji od tri osnovne datoteke za rad sa semantičkim webom, a to su: RDF/XML *parser* (raščlanjivač), *N-Triples* serijalizator i *Simple Model* klasa (omogućuje jednostavne metode za rad s opisima izvora podataka). *RDF/XML parser* služi za rasčlanjivanje i učitavanje podataka s lokalnog računala ili preko weba. *Parser* je moguće konfigurirati za dobavljanje sadržaja pretvorenog *RDF/XML*-a ili za zadržavanje čistog *RDF/XML*-a. *Parser* ne provjerava točnost rasčlanjenog koda kako bi potpuno zadovoljio *RDF* standard, već samo *XML* greške i neke osnovne dijelove *RDF/XML* sintakse. Rezultat procesa rada *parsera* je ili poruka o grešci ili niz *RDF* trojki.

Serijalizator generira *N-Triples* podatke iz niza trojki. Struktura procesuiranih trojki mora zadovoljavati strukturu *ARC* trojki. Svaki niz ulaznih podataka ima ključne dijelove *s* (subjekt čvora niza podatka), *p* (predikat *URI* podatak) i *o* (objekt čvora podatka). Serijalizator pruža mali broj konfiguracijskih opcija. Dopušta postavljanje razmaka i prebacivanje u novi red [Slika 63].

```
/* class inclusion */
include_once("ARC_ntriples_serializer.php");

/* class configuration (optional) */
$args=array("linebreak"=>"\n", "spacer"=>"\t");

/* instantiation */
$ser=new ARC_ntriples_serializer($args);

/* displaying the N-Triples code */
echo $ser->get_ntriples($triples);
```

Slika 63. Pokretanje i opcije serijalizatora

Jednostavni model klasa indeksira niz trojki i predlaže načine za povrat liste izvora podataka, za pronalaženje izvora podataka temeljem identifikatora, raščlanjivanje *RDF* listi podataka ili izvlačenje svojstvenih vrijednosti. Kombinacijom *RDF/XML parsera* i jednostavnog modela klasa pomoću *HTML-a* može se generirati pregled *RDF* podataka. Web aplikacija se sastoji od tri stranice. Prva stranica [Slika 64.] je naslovna i uz izbornik sadrži osnovne podatke o aplikaciji. Oblikovanje stranice rađeno je pomoću *HTML*-a i programa za pisanje *Notepad++*. Stranice su izrađene jednostavno bez kompliciranih sadržaja i služe izričito za pregled i pretraživanje izrađenih RDF bazi podataka, stoga je nepotrebno prezentirati *HTML* kod izrađene stranice.

Slika 64. Prva stranica aplikacije (naslovna)

Izbrnik sadrži tipke s kraticama triju stranica web aplikacije (Naslovna, *SPARQL* editor, *RDF* trojke). Izbornik je definiran u posebnoj datoteci *Header.php* [Slika 65.] koja se zatim poziva u svakoj *PHP* skripti posebno. Konačni izgled izbornika definiran je pomoću CSS-a u *Site.css*.

```
<ul id="menu">
<li><a href="Index.php">Naslovna</a></li>
<li><a href="endpoint.php">SPARQL editor</a></li>
<li><a href="podaci.php">RDF trojke</a></li>
</ul>
```

Slika 65. Prikaz koda datoteke Header.php

Druga stranica aplikacije uz izbornik sadrži i prozor za upisivanje *SPARQL* koda kojim se pretražuje *RDF* baza podataka. *SPARQL* kodom se može pretraživati ali i izmjenjivati baza podataka. Naredbom *INSERT* moguće je ubaciti neke nove *RDF* u bazu podataka, a naredbom *DELETE* obrisati ih. Uz lokalnu bazu, moguće je pretražiti i neke internet baze podataka npr. *Dbpedia*. Druga stranica pokreće se kraticom koja vodi na adresu *http://localhost/arc2-starter-pack/endpoint.php*. Datoteka *endpoint.php* osim podataka o bazi, korisniku i količini dotupnih rezultata sadrži i funkciju za učitavanje datoteke *config.php*.

Unutar *PHP*-a učitavanje skripte kao što je *config.php* izvršeno je pomoću naredbe *include_once*. Datoteka *endpoint.php* osim navedenog ne sadrži ništa relevantno, međutim, bitna je jer povezuje podatke o bazi podataka s *SPARQL endpoint*-om. Svaka datoteka koja prikazuje ili pretražuje *RDF* bazu podataka sadrži naredbu za učitavanje datoteke *config.php*. Dotična datoteka zatim poziva ostale datoteke u *ARC* direktoriju koje sadrže *SPARQLendpoint*, *parser*, serijalizator i druge potrebne funkcije. Uz mogućnost upisivanja *SPARQL* upita postoje i opcije za odabir formata rezultata pretraživanja. Rezultat se može dobiti ili u posebnoj *PHP* datoteci ili ispisom na trenutnoj stranici i to u oblicima: *XML*, *JSON*, serijalizirani *PHP*, *Turtle*, *RDF/XML*, *HTML* tablica i dr. Upit se izvršava pritiskom na tipku „Pošalji upit“. Tipkom „Reset“ cijeli upit se postavlja na unaprijed definirani upit koji je postavljen u skripti *ARC2_StoreEndpoint.php*. Pri izradi ovakvog tipa stranice moguće je postaviti još neke postavke kao što su postavljanje samo određene baze podataka koja će se upitom pretraživati. Moguće je i postaviti izbor između lokalne baze podataka ili internetske. U prozor za *SPARQL* upit kao početni upit je postavljen već objašnjeni upit za odabir obrazaca ponašanja kod ontologije robotskog sustava. Klikom miša na tipku „Pošalji upit“ dobiva se rezultat na istoj stranici ispod navedene tipke. Dobiveni rezultat moguće je pohraniti za kasniju upotrebu.

Slika 66. Druga stranica aplikacije (SPARQL editor)

Za *SPARQL* upit kod web aplikacije odabrana je druga verzija *SPARQL* koda iz razloga što *ARC* verzija *SPARQL*-a ne prepoznaje naredbu *BIND AS* koje je osnovni dio treće verzije *SPARQL* upita. Druga verzija *SPARQL* upita [Slika 67.] koja je korištena kod web aplikacije zbog razlike u sintaksi pisanja malo se razlikuje od upita kod korisničkog sučelja izrađenog *TopBraid Ensemble* alatom. *ARC* verzija ne prepoznaje uglate zagrade i skraćeni oblik pisanja (kao kod *Turtle* oblika *RDF*-a).

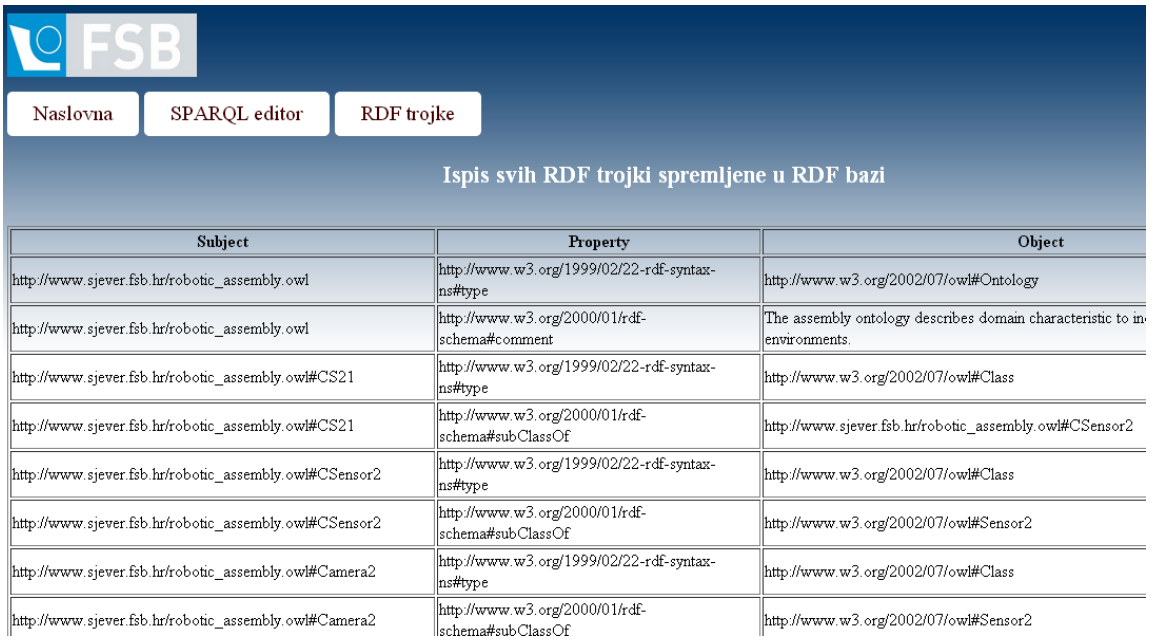
```
SELECT ?Behavioral_pattern ?BP
 WHERE {
 owl:tc11 owl:hasValue ?tc1 .
 owl:tc12 owl:hasValue ?tc2 .
 owl:cs11 owl:hasValue ?cs1 .
 owl:ob11 owl:hasValue ?ob1 .
 owl:vs11 owl:hasValue ?vs1 .
 owl:vs12 owl:hasValue ?vs2 .
 owl:sp11 owl:hasValue ?sp1 .

 ?BP owl:equivalentClass ?d .
 ?d owl:vrijednostTC1 ?tc1 .
 ?d owl:vrijednostTC2 ?tc2 .
 ?d owl:vrijednostCS1 ?cs1 .
 ?d owl:vrijednostOB1 ?ob1 .
 ?d owl:vrijednostVS1 ?vs1 .
 ?d owl:vrijednostVS2 ?vs2 .
 ?d owl:vrijednostSP1 ?sp1 .

 ?BP rdfs:label ?Behavioral_pattern }
```

Slika 67. Modificirani oblik druge verzije SPARQL koda

Treća stranica [Slika 69.] služi za prikaz svih RDF trojki učitanih u bazu podataka. Učitavanje podataka u MySQL bazu vrši se pomoću datoteke *load.php*. Postavljanjem direktorija odabrane datoteke u naredbu *LOAD* te upisivanjem adrese *http://localhost/arc2-starteck/load.php* se automatski sprema u željenu bazu podataka [Slika 68].

```
$store = ARC2::getStore($config);
if (!$store->isSetUp()) {
 $store->setUp();
}
$store->query('LOAD
<file:///C:\GotovPrimjer522.owl>');
```

Slika 68. Skripta za učitavanje podataka u MySQL bazu



Subject	Property	Object
http://www.sjever.fsb.hr/robotic_assembly.owl	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Ontology
http://www.sjever.fsb.hr/robotic_assembly.owl	http://www.w3.org/2000/01/rdf-schema#comment	The assembly ontology describes domain characteristic to in environments.
http://www.sjever.fsb.hr/robotic_assembly.owl#CS21	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
http://www.sjever.fsb.hr/robotic_assembly.owl#CS21	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.sjever.fsb.hr/robotic_assembly.owl#CSensor2
http://www.sjever.fsb.hr/robotic_assembly.owl#CSensor2	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
http://www.sjever.fsb.hr/robotic_assembly.owl#CSensor2	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2002/07/owl#Sensor2
http://www.sjever.fsb.hr/robotic_assembly.owl#Camera2	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
http://www.sjever.fsb.hr/robotic_assembly.owl#Camera2	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2002/07/owl#Sensor2

Slika 69. Treća stranica aplikacije (RDF trojke)

```
include_once("arc/ARC2.php");
include_once('config.php');
$store = ARC2::getStore($arc_config);
if (!$store->isSetUp()) {
 $store->setUp(); /* create MySQL tables */
}
$q = '
SELECT DISTINCT ?subject ?property ?object WHERE { ?subject ?property ?object .
}
';
$rows = $store->query($q, 'rows');
$r = '';
if ($rows = $store->query($q, 'rows')) {
$r = '<table border=1>
<th>Subject</th><th>Property</th><th>Object</th>'."\n";
foreach ($rows as $row) {
$r .= '<tr><td>'.$row['subject'] .
'</td><td>'.$row['property'] .
'</td><td>'.$row['object'] . '</td></tr>'."\n";
}
$r .='</table>'."\n";
}
else{
$r = '<em>No data returned</em>';
}
echo $r;
```

Slika 70. Prikaz skripte za ispis svih trojki untar baze podataka

8. ZAKLJUČAK

Trenutni web je zbog ogromne količine podataka došao do svojih granica i potreban mu je dodatni poticaj za daljnji razvoj. Semantički web ima namjeru srediti stanje te dopustiti aplikacijama, a ne samo ljudima, da aktivno sudjeluju u skupljanju, obradi i publiciranju sadržaja. U ovom radu proučene su i primjenjene tehologije za izradu, prikaz i eksploataciju povezanih podataka (eng. *Linked data*) koji čine osnovu semantičkog weba. Semantički opis sadrži ontologiju napisanu *OWL-om* koja se pokazala kao pogodan medij za pohranu, dijeljenje te ponovno korištenje znanja o domeni od interesa. Uzimajući u obzir razvijeni ontološki model i prikupljenog znanja pomoću senzora, ontološka jezgra može ponuditi jedno ili više rješenja u vidu, tzv. obrazci ponašanja (engl. *Behavioral Patterns*). obrasci ponašanja predstavljaju skup operacija koje robot može izvršavati kao odgovor na trenutno stanje u okolini. U slučaju da ontologija ponudi više mogućih rješenja (istovremeno dva ili više obrasca ponašanja), odluku o konačno odabranom algoritmu može se donijeti sustavom kao što je npr. *Bayesova* mreža. Prilikom razvoja ontološke jezgre nastojalo se da jezgra bude što jednostavnija kako bi model bio što pregledniji za kasnije održavanje, nadogradnju te implementaciju na realnim sustavima. Kako bi se postigla minimalna kompleksnost ontološke jezgre, dio logičkih operacija je prebačen na *SPARQL* jezik pomoću kojeg se šalju upiti ontološkoj jezgri. Uzimajći u obzir da web tehnologija kasni za razvojem u odnosu na moguću kompleksnost ontologija, navedeni pristup olakšava integraciju ontologije u sklopu web poslužitelja. Smatra se da podrška za kompleksnije ontologije još neko vrijeme neće biti zadovoljavajuća, zato valja težiti jednostavnosti i inovativnim rješenjima.

Pri izradi ontološkog modela vođeno je računa o kompatibilnosti ontološke jezgre modela s drugim sličnim ontologijama koje opisuju domene proizvodnih djelatnosti. Na temelju proučenih razvijenih ontologija i onih koje se trenutno razvijaju izrađena je ontološka jezgra modela prema uzoru na sustav robotske montaže u Laboratoriju za projektiranje izradbenih i montažnih sustava Fakulteta strojarstva i brodogradnje. Ontološka jezgra modela izrađena je *TopBraid Composer* alatom tvrtke *TopQuadrant* koji se pokazao kao jedan od boljih alata te namjene. Nažalost, alat *Composer* kao i cijeli paket alata *TopBraid Suite* nije besplatan, međutim, dostupan je u probnoj verziji od 30 dana. Odabran je zbog jednostavnosti pri korištenju i mogućnosti slanja SPARQL upita tijekom izgradnje ontološke jezgre čime se jednostavno može provjeriti funkcionalnost i mogućnosti željene ontološke jegre.

TopBraid Composer daje jednostavna rješenja za izradu ontološke jezgre, gotove datoteke moguće je spremati u više formata, kompatibilan je s trenutnim tehnologijama na tržištu i s ostatkom paketa *TopBraid Suite.*

Sveprisutno računarstvo omogućilo je integraciju računala (robota i drugih automatiziranih sustava) u okolinu koja se konstantno raznim senzorima propituje. Time se omogućuje stvaranje aplikacija koje su u ovisnosti o namjeni u manjoj ili većoj mjeri svjesne konteksta. Sustav koji svoje odluke temelji na kontekstualnoj spoznaji može se prilagoditi relativno jednostavno da radi različite zadatke u odnosu na klasične industrijske proizvodne linije. Znanje koje je integrirano u model je prenosivo, odnosno može biti dijeljeno, mijenjano te ponovno korišteno na različitim aplikacijama uz određene modifikacije. Korištenjem alata *TopBraid Ensemble* razvijeno je korisničko sučelje sa svim potrebnim funkcijama za pregledavanje i pretraživanje ontološkog modela. *TopBraid Ensemble* je dio spomenutog paketa alata *TopBraid Suite* i uz *Composer* i *Live Enterprise Server* čini cjelinu za izradu i spremanje i eksploataciju ontološkog modela. *Ensemble* je dostupan samo u starijim verzijama *TopBraid* paketa, za novije verzije još je u razvoju. Kao zamjena može se koristiti *SWP SPARQL Web Pages* koje nije grafičko sučelje kao i *Ensemble,* ali dozvoljava izgradnju identičnih aplikacija. U sklopu rada razvijena je i web aplikacija temeljena na *ARC2* sustavu, *PHP*-u i *MySQL*-u. Kombinacijom navedenih tehnologija razvijena je kvalitetna i jednostavna semantička aplikacija. *ARC2* daje kompletno rješenje koje sadrži serijalizatore i *parsere* te koristi *MySQL* bazu podataka kao *RDF* Store. Korištenjem *SPARQL endpointa* moguće je pretraživati razne web stranice, a podržava i neka naknada proširenja. Jedno je od najraširenijih sustava, ali nažalost, 2011. godine je prestao s razvojem. Unatoč tome sve njegove funkcije su usklađene sa zadnjim *W3C* standardima.

Trenutno osnovne tehnologije semantičkog weba nemaju punu funkcionalnost rada sa znanjem (zaključivanje) kao što to posjeduju neke ranije razvijene tehnologije. Semantički web je u razvoju i postoji još puno mjesta za poboljšanje, unaprijeđenje i razvoj novih standarda. Kao i kod svih novih tehnologija, konkretnih opisa i podataka o razvoju semantičkih aplikacija nema puno. Razvijeni sustav zaključivanja i predlaganja obrasca ponašanja funkcionira i uz web aplikaciju lako je primjenjiv. Implementacija semantičkog weba u postojeći sustav odvija se sporo, no razvojem semantičkih tehnologija očekuje sve veća primjena.

LITERATURA

[1] Sabou, M.: Building web service ontologies, Ph.D. thesis, Vrije Universiteit Amsterdam, the Netherlands, 2006

[2] *Youtube.com* statistika [Online] http://www.youtube.com/yt/press/statistics.html, [15.05.2013]

[3] *Facebook.com* statistka [Online] http://www.statisticbrain.com/facebook-statistics/ , [15.05.2013]

[4] Allemang, D., Hendler, J.:*Semantic web for working ontologiest*, Elsevier Amsterdam, 2011.

[5] *Internet* – Wikipedia [Online] http://hr.wikipedia.org/wiki/Internet, [15.05.2013]

[6] Stipančić, T.: *Spoznajni model upravljanja grupom industrijskih robota*, Ph.D. thesis, Fakultet strojarstva i brodogradnje, University of Zagreb, 2013

[7] Žagar, M. *Sveprisutno računarstvo*, Fakultet elektrotehnike i računarstva, University of Zagreb, 2013

[8] Want, R.: *Ubiquitous Computing Fundamentals*, J. Krumm, CRC Press, 2010

[9] *Semantika* –Wikipedia [Online] http://bs.wikipedia.org/wiki/Semantika, [25.05.2013]

[10] Slavić, A.: *Semantički Web, sustavi za organizaciju znanja i mrežni standardi*, University College London, 2004

[11] Dominigue, A., Fensel, D., Hendler, J.A.: *Handbook of Semantic Web Technologies*, Springer-Verlag Berlin Heidelberg, 2011

[12] Cundeković, M.: *Povezani podaci u novom ustroj interneta*, Fakultet strojarstva i brodogradnje, University of Zagreb, 2013

[13] Berners-Lee, T., Hendler J., Lassila, O.: *The Semantic Web*, Scientific American, 2001

[14] Paunović, L., Stokić A.: *Utjecaj ontologija na funkcionalnost weba*, Infoteh-Jahorina Vol.11, 2012

[15] Radovanović, D.:*Semantički web i elektronski izvori informacija*, Infoteka 4 vol. 2, 2003

[16] Legg, C.: *Ontologies on the semantic web*, University of Waikato, New Zeland, Annual Review of Information Science and Technology 41, 2007

[17] Verić, V., Đuraković, A.: *Semantički web*, Sveučilište Josipa Jurja Strossmayera u Osijeku, Odjel za matematiku, 2013

[18] Jurić, D.: *Metode učenja ontologija, trenutno stanje i problemi*, Fakultet elektrotehnike i računarstva, University of Zagreb, 2008

[19] Guarino, N.: *Formal Ontology in Informaton systems*, 1st international Conference on Formal Ontology in Information Systems (FOIS '98). Trento, Italy. IOS Press, Amsterdam.

[20] Horrocks, I., Patel-Schneider, P., Van Harmelen : From *SHIQ and RDF to OWL* : *the making of a Web ontology language*, Journal of web semantics, 2003

[21] Doleček, V., Karabegović, I: *Robotika*, Tehnički fakultet Bihać, 2002

[22] Jerbić, B., Nikolić G., Kunica, Z.: *Projektiranje automatskih montažnih sustava*, Školska knjiga, Zagreb, 1996

[23] Ungureanu, V., Honciuc, I.: *RDF based User Interfaces*, Computer Science Faculty, University of Iasi, Software Systems Engineering specialization

[24] Paulheim, H., Probst, F.: *Ontology-Enhanced User Interfaces*: A Survey, SAP Reasearch Center Darmstadt

[25] McIntyre, A., Durham, E.: *ARC RDF Store*, CSC 8711 Project 4, Topic report-Manual, Revision 2, 22.04.2011

[26] Sugumaran, V., Gulla, J.:*Applied Semantic Web Technologies*, CRC Press, Tylor & Francis Group, 2012

[27] Fensel, D., Hendler, J., Lieberman, H., Wahlster, W.: *Spinning the Seantic Web*, the MIT Press, Cambridge, Massachusetts, London, England, 2005

[28] TopQuadrant: *TopBraid Ensemble Application Development and Reference Guide*, relese 3.4,, 28.11.2010

[29] Zhou, J.: A *semantic framework for the delivery of e-government information and services*: The case of New Zeland, Thesis for the degree of Doctor of Philosophy, Information Systems, Massey University, Palmerston North, New Zeland, 2011

[30] Segaran, T., Evans, C., Taylor, J.: *Programing the Semantic web*, O'Reilly Media, Inc.,1005 Gravenstein Highway North, Sebastopol, CA 95472, 07.2009

[31] Matthews, B.: *Semantic Web Technologies*, CCLRC Rutherford Appleton Laboratory, JISC Technology and Standards Watch

[32] Kollia, I., Glimm, B., Horrocks I.: *Answering Queries over Ontologies with SPARQL*, ECE School, National Technical University of Athens, Greece, Oxford University Coputing Laboratory, UK, 2011

[33] Guo, Y., Pan, Z., Heflin, J.: *LUBM: A benchmark for OWL knowledge base systems*. J. Web Semantics 3, 2005

[34] *PHP Hypertext Preprocessor* - Wikipedia [Online]
https://en.wikipedia.org/wiki/PHP [15.06.2013]

[35] DuCharme, B.: *Learning SPARQL*, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Seastopol, CA 95472, 2011

[36] Harth, A., Janik, M., Staab, S.: *Semantic Web Architecture*, Karlsruhe Institute of Technology, University of Koblenz-Landau, 31.08.2010

[37] *Architecture of Linked Data Application*, Linked Data: Evolving the Web into a Global Data Space [Online]
http://linkeddatabook.com/editions/1.0/#htoc84 [15.06.2013]

[38] *ARC2 system*, ARC2 library [Online]
https://github.com/semsol/arc2/wiki/_pages [15.06.2013]

[39] Antoniou, G., Van Harmelen, F.: *A Semantic Web Primer*, The MIT Press, Cambridge, Massachusetts, London, England, Second Edition, 2008

[40] Want, R.,: *An introduction to Ubiquitous Computing*, Ubiquitous Computing Fundamentals, J. Krumm, Ed.: CRC Press, 2010

[41] Davis, M., Phillips, J.: *Learning PHP & MySQL*, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Seastopol, CA 95472, 2007

[42] Nowack, B.: *ARC: appmosphere RDF Classes for PHP Developers, Appmosphere Web Applications*, Kruppstr. 100, Essen, Germany, 2009

PRILOZI

I. CD-R disc